POSGRADO INTERINSTITUCIONAL DE CIENCIA Y TECNOLOGÍA

SISTEMA ELECTRÓNICO DE ADQUISICIÓN Y PROCESAMIENTO DE SEÑALES CON COMUNICACIÓN USB

TESIS

QUE PARA OBTENER EL GRADO

ACADÉMICO DE

MAESTRO EN CIENCIA Y TECNOLOGÍA

EN LA ESPECIALIDAD DE MECATRÓNICA

PRESENTA

Ing. Hiram Abif Hernández Rivera

DIRIGIDA POR EL

Dr. Jesús Carlos Pedraza Ortega

SANTIAGO DE QUERÉTARO, QRO., JULIO DEL 2016

POSGRADO INTERINSTITUCIONAL
CIENCIA Y TECNOLOGÍA

Director de Posgrado
PICYT – CIDESI
Querétaro

Los abajo firmantes, miembros del Comité Tutorial del alumno **Hiram Abif Hernández Rivera**, una vez leída y revisada la Tesis titulada **"SISTEMA ELECTRÓNICO DE ADQUISICIÓN Y PROCESAMIENTO DE SEÑALES CON COMUNICACIÓN USB"**, aceptamos que la referida tesis revisada y corregida sea presentada por el alumno para aspirar al grado de **Maestría en Ciencia y Tecnología** en la opción terminal de **Mecatrónica** durante el Examen de Grado correspondiente.

Y para que así conste firmo la presente a los 21 días del mes de Junio del año dos mil dieciséis.

Dr. Jesús Carlos Pedraza Ortega
Tutor Académico

CIENCIA Y TECNOLOGÍA

Director de Posgrado
PICYT – CIDESI
Querétaro

Los abajo firmantes, miembros del Jurado del Examen de Grado del alumno **Hiram Abif Hernández Rivera**, una vez leída y revisada la Tesis titulada **"SISTEMA ELECTRÓNICO DE ADQUISICIÓN Y PROCESAMIENTO DE SEÑALES CON COMUNICACIÓN USB"**, aceptamos que la referida tesis revisada y corregida sea presentada por el alumno para aspirar al grado de **Maestría en Ciencia y Tecnología** en la opción terminal de **Mecatrónica** durante el Examen de Grado correspondiente.

Y para que así conste firmamos la presente a los 21 días del mes de Junio del año dos mil dieciséis.

Dr. Jesús Carlos Pedraza Ortega
Presidente

Dr. Hugo Jiménez Hernández
Secretario

Dr. Leonardo Barriga Rodríguez
Vocal

RESUMEN

El desarrollo de sistemas electrónicos utilizando procesadores de señales digitales ha ido aumentando paulatinamente, debido a la necesidad de manejar y procesar grandes cantidades de datos en tiempo real. Un ejemplo del uso de estos sistemas se expondrá en el presente trabajo, el cuál será implementado para el procesamiento de señales bioeléctricas. El desarrollo consiste en un algoritmo para adquirir, filtrar y enviar datos, hacia una computadora personal desde un procesador de señales digitales. La comunicación entre el procesador de señales digitales y la computadora se efectúa mediante el puerto serial universal, configurado para transferencias de control, isócronas y de interrupción. El trabajo aquí desarrollado es aplicado para la adquisición y comunicación entre un Electroencefalógrafo Digital y una computadora. Es importante señalar que el algoritmo queda abierto para usarlo en cualquier otro dispositivo que requiera de una comunicación hacia una computadora y en los cuales se tenga como requerimiento crítico el tiempo de envío de los datos. Como resultado de este trabajo, se ha podido sustituir una tarjeta de adquisición de datos comercial. Logrando con esto el aumento de competitividad y reducción de costos entre equipos médicos desarrollados en el país y aquellos desarrollados en el extranjero.

ABSTRACT

Development of electronic systems using Digital Signal Processors has been increased gradually, due to the necessity of handling and processing a great amount of data in real time. An example of this will be exposed on this work, which will be implemented in the processing of bioelectrical signals. The development consists of an algorithm to collect, filter and send data, towards a personal computer from a digital signal processor. The communication between digital signal processor and the computer uses the universal serial bus, setting for control, isochronous and interrupt transfers. The work developed here is applied for the acquisition and communication between a Digital Electroencephalogram and a computer. It is very important to notice that the proposed algorithm could be used in any other device that requires of a communication towards a computer and in which has critical requirement like the time of data sending. The result of this work, it has been possible to replace a data acquisition commercial board. Therefore, the competitiveness could be increased and the reduction of costs between medical equipment developed in the country and those developed abroad can be reduced.

TABLA DE CONTENIDOS

TABLA DE FIGURAS

CAPÍTULO 1 - INTRODUCCIÓN

1.1 ANTECEDENTES

México a través de los años ha estado impulsado el desarrollo de tecnología propia mediante el trabajo en conjunto con la investigación, para así alcanzar el objetivo propuesto en los planes de desarrollo del país, reducir la exportación de productos [1].

Hablando de la industria médica, en la historia reciente, en el país han existido muy pocas empresas las cuales se dedicaban al diseño y desarrollo de equipos médicos de diagnóstico en México, esto implica que se tuviera que importar cada equipo médico y logrando así los altos costos de dichos equipos, por lo cual la secretaría de salud y hospitales privados han invertido una fuerte cantidad de dinero para conseguir los aparatos necesarios para su infraestructura actual [2].

Así fue como en el 2003 el Centro de Ingeniería y Desarrollo Industrial (CIDESI) comenzó a trabajar en el desarrollo de equipos médicos, enfocándose en el área de neurología y actualmente en el área de cardiología, ver figura 1.1.

Figura 1.1 Electroencefalógrafo Digital y Monitor Cardíaco Inalámbrico desarrollados por CIDESI.

El presente trabajo contribuye al desarrollo de equipos médicos de diagnóstico en el país, ya que en primera instancia será aplicado hacia un electroencefalógrafo (EEG) digital desarrollado por personal del CIDESI®, cabe mencionar que este es uno de los primeros equipos médicos de diagnóstico diseñados y desarrollados completamente en nuestro país y el primero de este tipo.

El EEG digital desarrollado ha utilizado una tarjeta de adquisición de datos comercial conectada a un puerto PCI de una PC para la adquisición de las señales bioeléctricas provenientes de un sistema de acondicionamiento, llamado amplificador, ver figura 1.2.

El amplificador del EEG es un equipo electrónico, al cual se le conectan 24 sensores, llamados electrodos, los cuales llevan las señales provenientes del cuero cabelludo

hacia el amplificador, dentro de este equipo las 24 señales bioeléctricas son acondicionadas para poder trabajar con ellas. El acondicionamiento de las señales es la realización de diferentes etapas de amplificación y filtrado analógico de las mismas.

Actualmente la computadora, con la cual se comunica el amplificador es un componente principal en el equipo, debido a la cantidad de tareas que en ella se ejecutan para cumplir con la adquisición y con las diferentes herramientas que cuenta el EEG digital de CIDESI® para el procesamiento de las señales de electroencefalografía.

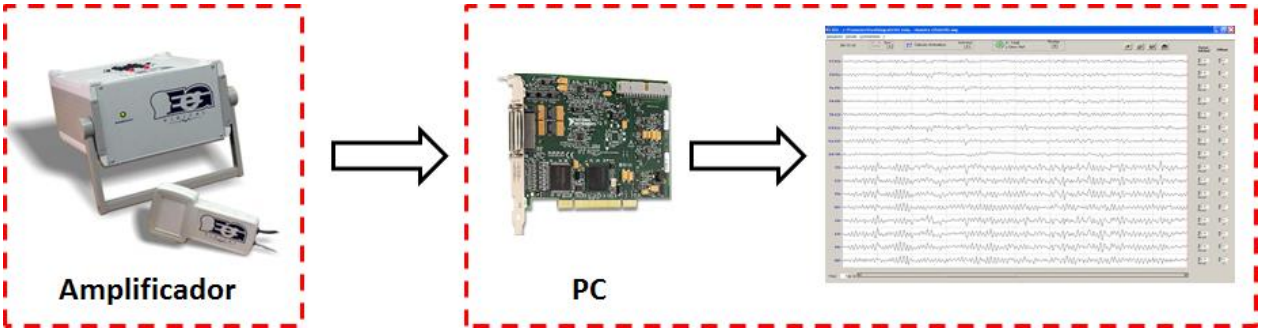


Figura 1.2 Forma actual de comunicación del EEG Digital con una PC.

1.2 PLANTEAMIENTO DEL PROBLEMA

A continuación se mencionan las principales tareas que la PC del EEG Digital desarrollado tiene que realizar actualmente en línea, es decir mientras se está llevando a cabo un estudio de electroencefalografía:

1. Adquisición
2. Filtrado,
3. Generación de montajes bipolares
4. Almacenamiento de las señales bioeléctricas del cerebro.

Además si el usuario requiere de la herramienta de Video EEG, sincronizado con las señales del cerebro; entonces a las tareas antes mencionadas se adicionan la adquisición y procesamiento de video digital para lograr la sincronía y el almacenamiento del mismo.

Por todo lo anterior, la complejidad en el proceso de análisis en tiempo real de la señal de un EEG resulta complicada de manejarse en una PC, por lo que una alternativa útil

es contar con determinismo, para realizar cada una de las tareas necesarias en un tiempo definido. Lo anterior nos lleva a que la PC debiera ser lo más potente posible y con un sistema operativo determinístico, tiempo real, para lograr la ejecución de tareas en tiempo y con todo esto el costo del EEG se incrementa de manera considerable.

Durante la venta de este equipo médico se observa que el usuario la mayoría de las veces ya cuenta con un equipo de cómputo, el cual pocas veces cumple con los requisitos necesarios para garantizar el correcto funcionamiento del EEG. Lo anterior incluye que el usuario requiere utilizar el EEG con una computadora portátil, pero como mencionamos anteriormente el equipo es comunicado con una tarjeta de adquisición mediante un puerto PCI y estas computadoras portátiles no cuentan con este puerto de comunicación con lo cual se imposibilita el uso del EEG Digital de CIDESI® con una laptop.

Por todo lo anterior en este proyecto se plantea una posible solución a cuatro factores con los que cuenta el EEG Digital:

1. Reducción de procesos que actualmente realiza una PC con el EEG Digital.
2. Determinismo necesario (tiempo real)
3. Comunicación con cualquier computadora personal.
4. Disminución de costos.

Para cumplir con estos 4 objetivos se propone utilizar un Procesador Digital de Señales, DSP por sus siglas en inglés, el cual ayudará a reducir los procesos realizados en la PC, principalmente la adquisición de las señales y filtrado de las mismas, y así cumplir también con el determinismo necesario para un estudio del EEG. Además de utilizar el puerto serial universal, USB, para la comunicación entre el EEG digital y cualquier equipo de cómputo actual y futuro.

Al lograr lo anterior el EEG Digital podrá ser un equipo de diagnóstico médico más competitivo contra equipos similares importados, reduciendo su costo y mejorando algunas características. Como es el caso de equipos de los siguientes fabricantes: Neurotec, GE Medical Systems, Phillips Medical, entre otros.

1.3 JUSTIFICACIÓN

En forma general la justificación principal de este desarrollo es el generar mayor competitividad entre los equipos desarrollados por CIDESI® en comparación con aquellos importados. La intención de este desarrollo es que sea un módulo el cual sea aplicable no solo al EEG si no más adelante a cualquier otro equipo médico desarrollado en CIDESI®, pero para este trabajo solamente será enfocado al EEG Digital.

Como ya se comentó anteriormente, desarrollando un sistema propio de adquisición de señales bioeléctricas se puede reducir los costos del EEG al ya no depender de una tarjeta comercial desarrollada por una compañía externa.

A continuación podrían existir las siguientes interrogantes: ¿Por qué el uso de un DSP? y ¿Por qué el uso del puerto USB?

Para la primera interrogante podremos decir, que este procesador es dedicado al procesamiento de las señales que si bien es cierto por el momento sólo tendremos el filtrado digital de las señales bioeléctricas pero más adelante necesitaremos agregar más procesamiento de la señal en el mismo sistema de adquisición y no en la PC, como sería la identificación de la morfología de las señales bioeléctricas en línea, entre otras más, existen trabajos actuales los cuales donde se demuestra este interés en otras parte del mundo [3], [4], [5], [6], [7].

Además que el DSP es un dispositivo de muy bajo consumo de energía y de un tamaño muy pequeño, el cual realiza operaciones matemáticas en tiempo real y cuenta con un BIOS integrado para realizar la calendarización de las tareas a realizar en tiempos determinados.

Para la segunda interrogante la respuesta sería que gracias a este puerto de comunicación podemos lograr hasta un máximo de 12 Mb/s para la transferencia de información, usando el dispositivo seleccionado, y hasta 5Gbps cambiando de dispositivo a uno que soporte la especificación 3.0.

Además de ser un protocolo de comunicación estándar con alta confiabilidad, facilidad de conexión y cada vez más usual en cualquier sistema de cómputo. Así como también

sigue habiendo cada vez más investigación en la mejora de esta tasa de transferencia [8].

Existen trabajos relacionados donde se ve el interés en equipos médico de diagnóstico el utilizar procesamiento de señales digitales, como se puede ver en [9], [10], [11] y [12], pero a diferencia del trabajo propuesto, los mencionados anteriormente algunos son solamente el algoritmo implementado en la PC y otros los cuales se implementa los algoritmos en un DSP, utilizan una tarjeta de comunicación con la PC.

1.4 OBJETIVOS

El Objetivo general planteado para este desarrollo es el siguiente:

Desarrollo e implementación de un sistema electrónico de propósito específico, que utilice el protocolo de comunicación USB e implemente un conjunto de operaciones sobre el EEG en un DSP para optimizar el uso de recursos en aplicaciones en línea.

Los objetivos específicos planteados para el desarrollo de este trabajo son los siguientes:

- Desarrollo de software en un DSP para la adquisición de 21 señales analógicas, provenientes del amplificador del EEG con una frecuencia de muestreo de hasta 1000Hz.
- Desarrollo del algoritmo para el filtrado digital de las señales en el propio DSP.
- Desarrollo de software para la comunicación entre el DSP y la PC mediante el puerto USB.
- Sincronizar la adquisición de datos con la comunicación USB.
- Modificar el software actual del EEG para integrar la comunicación con la tarjeta de adquisición de datos desarrollada en el proyecto.

En la figura 1.3 se muestra el concepto propuesto de solución para el presente trabajo.

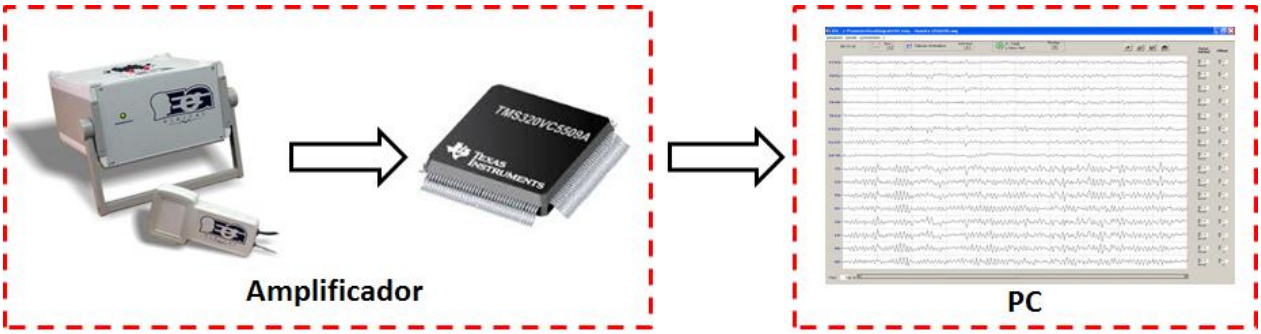


Figura 1.3 Modificación de la comunicación entre el EEG Digital y la PC utilizando un DSP.

1.5 ORGANIZACIÓN DE LA TESIS

La organización de este trabajo se presenta en 4 capítulos y 1 anexo. A continuación se menciona el alcance de cada capítulo.

En el capítulo 1 se presentó las generalidades y direcciones de la tesis, en donde se tiene una introducción para tener una idea general del proyecto y conocer las consideraciones que se tomaron para llevarlo a cabo.

En el capítulo 2 se presentan los fundamentos biomédicos y de procesamiento de señales, enfocándose en la introducción del funcionamiento de un Electroencefalógrafo y sus diferentes análisis que se realizan con estas señales, así como también se presentan los fundamentos del procesamiento de señales digitales, en el cual se realiza una comparativa entre las señales análogas y digitales, y se realiza una introducción al análisis espectral de las señales, ya que para el filtrado digital se necesitan de los conceptos de Transformada de Fourier y convolución.

En el capítulo 3 se presentan los conceptos tecnológicos utilizados en este desarrollo. Primeramente se dará una introducción a los Procesadores de Señales Digitales, DSP por sus siglas en inglés. También se mencionarán los fundamentos del puerto USB, las bases para el desarrollo de dispositivos con este tipo de comunicación, explicación de cada una de las diferentes transferencias que se pueden realizar mediante USB, se explica también el proceso de enumeración de un dispositivo, además la definición de las diferentes clases que existen de estos dispositivos y finalmente la comunicación hacia el host.

En el capítulo 4 se menciona el desarrollo del proyecto, tanto de la parte de hardware y un esquema general del desarrollo de software en el DSP y la PC. En este capítulo se presentan los resultados obtenidos en este trabajo.

En el capítulo 5, se exponen las conclusiones que se obtuvieron durante el desarrollo de este trabajo así como las perspectivas de trabajos futuros en los cuales este desarrollo sea parte fundamental.

Por último se muestran las Referencias usadas durante el desarrollo de este trabajo de tesis y el Anexo A correspondiente a una publicación en una conferencia nacional como requisito de titulación.

CAPÍTULO 2 - FUNDAMENTOS

2.1 FUNDAMENTOS BIOMÉDICOS

La ingeniería biomédica es una disciplina relativamente joven, y se ha convertido en un campo interdisciplinario muy importante. Los ingenieros biomédicos están involucrados en el diseño, desarrollo y utilización de materiales, dispositivos (como marcapasos, electros, etc.) y técnicas (tales como procesamiento de señales, inteligencia artificial, etc.) para la investigación y uso clínico. Estos ingenieros buscan nuevas soluciones para los diferentes problemas de la salud que confronta nuestra sociedad [1], [2], [13], desarrollando aplicaciones biomédicas, ver figura 2.1.

Figura 2.1 Aplicaciones biomédicas.

En la figura 2.1, se muestran dos aplicaciones biomédicas claras que están en desarrollo actualmente. La imagen de la izquierda se refiere al desarrollo de una aplicación de software la cual ayuda a los neurólogos a encontrar algún problema neurológico en un paciente, mostrando con un punto color rojo la zona afectada. La imagen de la derecha nos muestra una pequeña capsula que al ser administrada vía oral a un paciente, realizará un escaneo de los signos vitales por un periodo de tiempo establecido y al paso del mismo está cápsula será absorbida por el organismo.

Los cambios que han ocurrido en la medicina originados por los rápidos desarrollos en las ciencias aplicadas como son: química, física, ingeniería, fisiología, entre otras; han beneficiado un ambiente en el cual la investigación médica sea capaz de lograr grandes avances en el desarrollo de técnicas para el diagnóstico y tratamiento de enfermedades.

El campo de la ingeniería biomédica actualmente incluye muchas nuevas áreas, algunas de ellas incluyen:

1. Análisis de sistemas de ingeniería para problemas biológicos.
2. Detección, medición y monitoreo de señales fisiológicas (biosensores e instrumentación médica).
3. Interpretación de diagnóstico mediante técnicas de procesamiento de señal de datos bioeléctricos.
4. Dispositivos y procedimientos terapéuticos y de rehabilitación.
5. Dispositivos para remplazo o aumento de funciones corporales (órganos artificiales).
6. Análisis de computadora de datos relacionados con los pacientes y decisiones clínicas (informática médica e inteligencia artificial).
7. Imagen médica, por ejemplo, desplegar gráficas de un detalle anatómico o función fisiológica.
8. La creación de nuevos productos biológicos (biotecnología).

La ingeniería biomédica es una rama interdisciplinaria de la ingeniería que abarca desde proyectos teóricos, no experimentales, hasta aplicaciones en el “estado del arte” [14]. Estas pueden abarcar la investigación, desarrollo, implementación y operación. Un gran beneficio del uso de la ingeniería biomédica es que gracias a ésta se puede identificar los problemas y necesidades de nuestro actual sistema al cuidado de la salud, y que pueden ser resueltos usando tecnología de ingeniería y metodología de sistemas existentes. Consecuentemente, el campo de la ingeniería biomédica ofrece esperanza en la batalla continua para ofrecer un cuidado de la salud de alta calidad a un costo razonable.

2.2 PRINCIPIOS DE ELECTROENCEFALOGRAFÍA

La electroencefalografía es el registro y evaluación de los potenciales eléctricos (generalmente menores a 400uV) generados por el cerebro y obtenidos por medio de electrodos situados sobre la superficie del cuero cabelludo. El electroencefalograma (EEG) es el registro de la actividad eléctrica de las neuronas del encéfalo. Dicho registro posee formas muy complejas que varían mucho con la localización de los

electrodos y entre individuos. Un electroencefalógrafo es aquel equipo de médico de diagnóstico que se encarga de realizar los electroencefalogramas, mediante la adquisición y procesamiento de las señales del cerebro.

2.2.1 Perspectiva histórica

En 1870, Fritsch y Hitzig [14], observaron que al estimular, mediante corriente galvánica, determinadas áreas laterales de cerebros descubiertos se producían movimientos en el lado opuesto del cuerpo. En 1913, Prawdwicz-Neminski registró lo que llamó “electrocerebrograma” de un perro, siendo el primero en intentar clasificar semejantes observaciones. Es importante señalar que todos estos experimentos fueron realizados sobre cerebros descubiertos, debido a que las diferencias de potenciales son muy pequeñas y no existía aún alguna manera de amplificar estos voltajes, por lo que era imposible registrar los impulsos que alcanzaran el exterior del cráneo.

Fue hasta 1928 cuando Hans Berger ideó un método que prometía una investigación de la actividad eléctrica cerebral, descubriendo lo que se conoció como “ritmo de Berger”. Berger observó por ejemplo que cuando el sujeto abría los ojos o resolvía algún problema mentalmente se alteraba el ritmo amplio y regular, para estas fechas ya era posible el uso de amplificadores para el monitoreo de los potenciales eléctricos desde el cuero cabelludo.

2.2.2 Técnicas para grabar un EEG.

Las grabaciones de un EEG, desde el cuero cabelludo de la actividad neuronal del cerebro, permiten la medición de cambios de potencial en el tiempo entre un electrodo de señal y uno de referencia. Una etapa de importancia es el acondicionamiento, ya que por la naturaleza de los biopotenciales del cerebro, se presenta mucha susceptibilidad a interferencias de todo tipo, eléctricas e inclusive de movimiento muscular [15], [16]. Las señales del cerebro se encuentran en un rango de voltaje de entre 1 a 200uV y en un rango de frecuencia entre 0.5 a 100 hz.

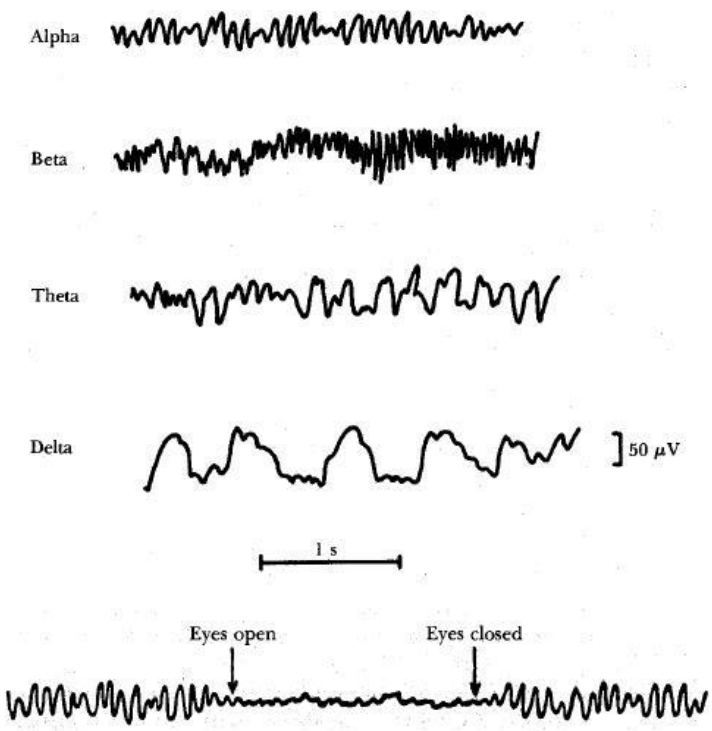


Figura 2.2 Ritmos normales en electroencefalografía.

La actividad bioeléctrica cerebral puede captarse por diversos procedimientos, sobre el cuero cabelludo, en la base del cráneo y/o en el cerebro expuesto; por lo que para captar la señal se utilizan diferentes tipos de electrodos, tales como: superficiales, basales y quirúrgicos.

En la figura 2.2 se muestran los ritmos normales que se presentan en un estudio de EEG, los cuales provienen de la corteza cerebral.

En el caso del EEG Digital desarrollado por CIDESI®, se adquieren 21 señales provenientes del cuero cabelludo más un punto de referencia que normalmente es ubicado en la frente, para así obtener el estándar internacional 10-20 para la colocación de los electrodos, en la figura 2.3 se muestra un diagrama con la ubicación y nombre de cada una de las posiciones de los electrodos bajo este estándar.

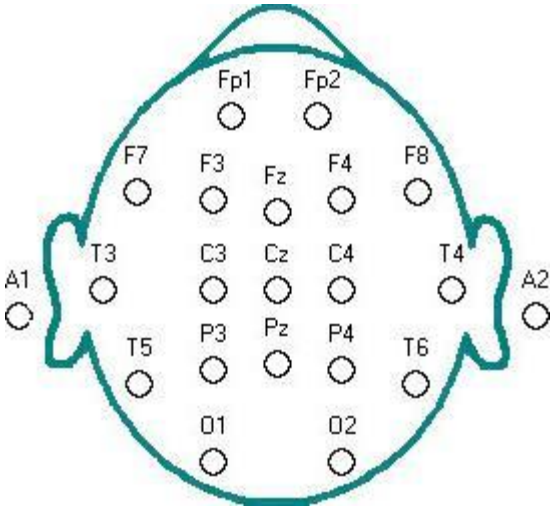


Figura 2.3 Nombre y ubicación de cada electrodo según estándar 10-20 para un EEG.

Como se mencionó anteriormente los potenciales bioeléctricos del cerebro tienen un rango de frecuencia que normalmente se encuentran entre 0.5 a 100hz, para casos clínicos, pero en casos especiales, por ejemplo, para estudios científicos se consideran de interés frecuencias muy superiores a éstas, por lo que se recomienda que cualquier equipo de electroencefalografía por mínimo sea capaz de muestrear señales de hasta 200hz y hoy en día se maneja un estándar de 1000Hz por canal monitoreado.

Normalmente se utiliza en la grabación del EEG montajes o derivaciones bipolares para la visualización de señales, lo cual lleva a la realización de una diferencia entre dos electrodos para formar un canal de visualización (Fp1-F7) y para algunas condiciones es importante obtener en modo referencial las señales, esto es que un electrodo menos la referencia se presenta en un canal visual (Fz-Ref).

A continuación se presenta una tabla con la subdivisión de frecuencias presentes en un electroencefalograma de acuerdo a la frecuencia predominante en la actividad actual del paciente.

Tipo de onda y voltajes	Frecuencia	Situación Mental
DELTA 10-50 µV	0.5-3.5 Hz	Estado hipnótico, hemisferio cerebral derecho en plena actividad, sueño profundo y meditación.
THETA 50-100 µV	3.5-7.5 Hz	Estado de vigilia, equilibrio entre los hemisferios izquierdo y derecho, plenitud y armonía.
ALFA	7.5-13 Hz	Relajación, tranquilidad, creatividad,

100-150 µV		inicio de actividad plena del hemisferio izquierdo y desconexión del hemisferio derecho.
BETA 150-200 µV	13-40 Hz	Estado de alerta máxima, es la situación normal cuando estamos despiertos, conduciendo, o trabajando en donde estamos en estado de alerta, ansiedad.

Tabla 2.1 Frecuencia dominante según actividad o condición del paciente [15].

Por lo anterior se puede observar que para el análisis de un EEG es indispensable identificar la frecuencia predominante sobre ciertas actividades y/o condiciones del paciente, es por esto que ciertas herramientas para el análisis de la frecuencia de los potenciales bioeléctricos son indispensables para el médico, como son: histogramas y/o mapeo cerebral.

El EEG Digital de CIDESI® cuenta con las siguientes características:

- Adquisición de 21 canales.
- Frecuencia de muestreo de hasta 1000Hz.
- 12 bits de resolución.
- Filtrado digital.
- Montajes configurados por el usuario.
- Mapeo cerebral.
- Histogramas de frecuencia.
- Video EEG.

2.3 SISTEMAS DE MEDICIÓN IDEAL DE BIOSEÑALES

Una representación esquemática de un sistema de medición biomédico típico es mostrada en la figura 2.4. El proceso fisiológico de interés es convertido a señal eléctrica vía un transductor. Un procesamiento de señal análoga es usualmente requerido, incluyendo amplificación y filtrado pasabajos o pasabanda. Debido a que es más fácil realizar el procesamiento de la señal utilizando métodos digitales, la señal análoga es convertida a digital usando un convertidor análogo a digital.

Una vez convertida, la señal es frecuentemente almacenada en memoria para facilitar el procesamiento subsecuente de la señal.

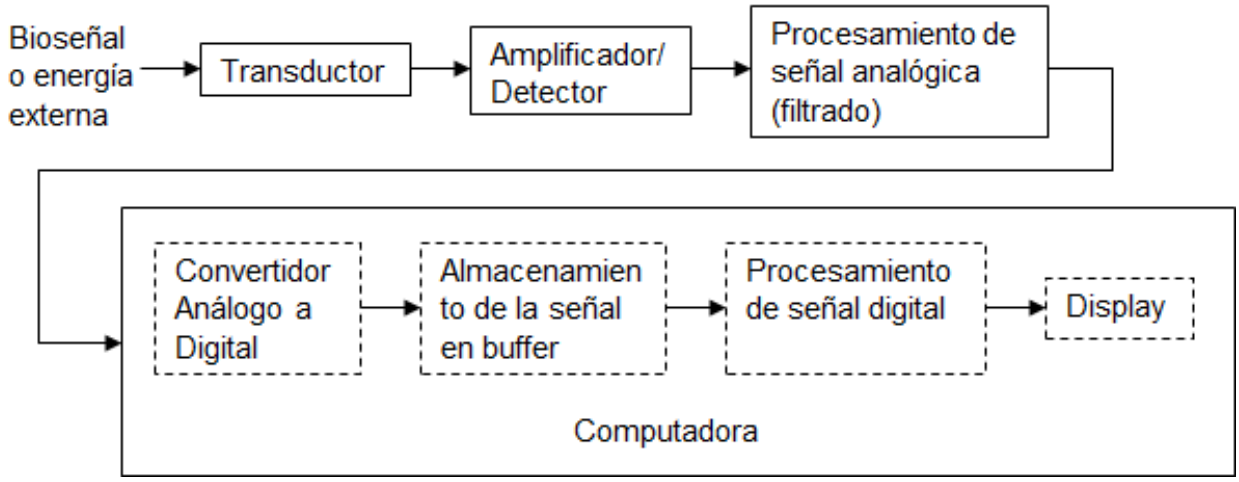


Figura 2.4 Representación esquemática de un sistema de medición de bioingeniería típico.

En aplicaciones de tiempo real los datos de entrada deberán ser procesados tan rápido, y en forma determinística como sea posible con un sistema de buffers mínimo, para no tener retrasos en la obtención de los datos.

En el presente trabajo se manejan los datos en tiempo real. El convertidor análogo a digital, el buffer de la señal y el procesamiento de ésta se realizará en base a un procesador digital de señales (DSP) de la empresa Texas Instruments, la selección de esta compañía se debe principalmente a que CIDESI es casa de diseño de esta empresa y por lo tanto se cuenta con un amplio expertis con las herramientas de desarrollo y contamos con línea directa a soporte técnico, lo cual nos permite disminuir algún riesgo en el desarrollo del trabajo aquí planteado.

El ruido o variabilidad de la señal es inherente en la mayoría de los sistemas de medición y frecuentemente es un factor limitante en el funcionamiento de un equipo médico. En las mediciones biomédicas, la variabilidad tiene 4 diferentes orígenes:

- Variabilidad fisiológica.
- Ruido ambiental o interferencia.
- Artefacto del transductor.
- Ruido electrónico.

La variabilidad fisiológica es debido al hecho de que la información que se desea está basada en un sujeto de medición a influencias biológicas diferentes a las de interés.

El ruido ambiental puede venir de fuentes externas o internas al cuerpo, un ejemplo clásico es el EEG, en el cual son captadas las señales provenientes del movimiento muscular, de los ojos y en determinados casos del corazón [4], [17].

Los artefactos en el transductor son producidos cuando el transductor responde a otras energías diferentes a las que se desean. Por ejemplo, grabaciones de potenciales eléctricos usando electrodos colocados sobre la piel son afectados por artefacto de movimiento, donde los electrodos responden tanto al movimiento mecánico como a la señal eléctrica deseada.

A diferencia de otras fuentes de variabilidad, el ruido electrónico tiene características y fuentes bien conocidas. Este tipo de ruido tiene un amplio rango de frecuencias que va desde DC hasta 10^{12} - 10^{13} Hz. También es llamado "ruido blanco", debido a que contiene energía en todas las frecuencias de interés para las aplicaciones biomédicas. La energía de este ruido es casi constante a lo largo de todo el rango espectral y se encuentra presente en cualquier sistema electrónico.

Por último se muestra en la tabla 2.3 las diferentes fuentes de ruido o variabilidad con sus respectivas causas y posibles remedios.

Fuente	Causa	Solución potencial
Variabilidad fisiológica	Medición indirectamente relacionada a la variable de interés.	Modificar enfoque general.
Ambiental (interno o externo)	Otras fuentes de energía de forma similar.	Cancelación de ruido, diseño del transductor.
Artefacto	Respuesta del transductor a otras fuentes de energía.	Diseño del transductor.
Electrónico	Ruido térmico o de disparo.	Transductor o diseño electrónico

Tabla 2.2 Fuentes de ruido en sistemas biomédicos [14].

Un estudio de electroencefalografía es afectado por el ruido ambiental, artefacto y ruido electrónico, por lo que en la parte analógica cuenta con amplificación y filtrado analógico el cual es ajustado para obtener señales máximas de ±3.0v para este trabajo, ya que actualmente maneja ±10.0v máximo antes de entrar a la parte de adquisición de datos.

2.4 PROCESAMIENTO DIGITAL DE SEÑALES

A continuación se explicarán algunos de conceptos básicos del procesamiento digital de señales, ya que es uno de los temas principales de este trabajo, se explicarán las diferencias entre señal analógica y digital, y por último se aborda el tema de filtros digitales.

El procesamiento digital de señales utiliza información codificada en forma de secuencia de mediciones numéricas. En la mayoría de los casos, estas señales provienen de datos monitoreados desde el mundo real: Biopotenciales, vibraciones sísmicas, imágenes, ondas del sonido, etc.

El procesamiento digital es la matemática, el algoritmo y las técnicas usadas para manipular estas señales después de que hayan sido convertidas en su forma digital [18].

El procesamiento digital de señales ha sido muy usual en la medición y análisis de las señales en dos maneras diferentes. La primera es pre-acondicionar la señal medida rechazando el ruido e interferencia [19], [20], [21], [22], como en los filtros digitales, y la segunda es interpretar las propiedades de los datos obtenidos, [23], [24], [25], [26], como en la identificación de la morfología de una señal bioeléctrica o en la identificación de objetos.

Los filtros digitales pueden ser encontrados comúnmente en equipos electrónicos médicos como electrocardiógrafos (ECG) [19], [22], [27], y electroencefalógrafos (EEG) [5], [6], [17] los cuales graban los biopotenciales, señales débiles, con la presencia de gran cantidad de ruido e interferencia.

2.5 SEÑALES CONTINUAS

Una señal que puede tomar una cantidad de valores infinitos y en la cual el tiempo puede ser divido en incrementos infinitamente pequeños es una señal *continua en amplitud* y *continua en tiempo,* hoy en día esta señal es llamada *analógica*, ver figura 2.5.

Ahora bien, si sólo se presenta la señal en instantes de tiempo específicos, pero la amplitud toma cualquier valor, ésta es *continua en amplitud*, pero *discreta en tiempo*.

Hay un tercer tipo de señal la cual es continua en todo el tiempo, pero solamente puede tener ciertos valores en su amplitud. Tal señal es descrita como *discreta en amplitud* y *continua en tiempo*.

El cuarto tipo de señal es aquella que está definida en instantes de tiempo específicos y que solamente puede tener ciertos valores ya definidos, esta es descrita como *discreta en amplitud* y *discreta en tiempo* y la cual es llamada señal *digital*, ver figura 2.5.

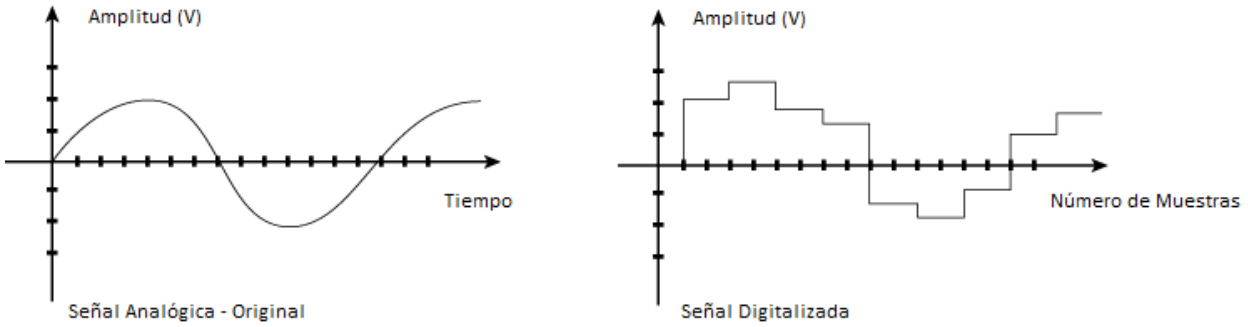


Figura 2.5 Señal analógica vs Señal digital.

Los potenciales bioeléctricos del cerebro, corazón, etc. son entonces de acuerdo a las definiciones anteriores señales analógicas. En este trabajo el procesamiento se realizará en un DSP y de ahí se enviará la información hacia una PC, por lo que es necesario llevar a cabo una conversión analógica a digital de la señal. A continuación se dará una explicación teórica acerca del proceso de conversión analógica a digital y la implementación en este trabajo.

2.6 CONVERSIÓN ANALÓGICA A DIGITAL

En el presente trabajo se utilizan dos convertidores análogo-digital, TLV2556 [43], los cuales tienen 12 bits de resolución, 200 KSps, 11 canales de entrada analógica, con los cuales podemos cumplir con el requisito de monitorear las 21 señales bioeléctricas del cerebro.

La señal analógica es convertida a digital mediante un proceso llamado digitalización o conversión análoga-digital, el cual a su vez depende de dos operaciones: *muestreo*, el

cual se encarga de tomar las muestras de la señal en un tiempo específico y en forma regular. Y la segunda operación es la *cuantificación* que consiste en aproximar valores continuos de la amplitud en un grupo de valores manejables en el sistema digital.

En la figura 2.6 se muestran los elementos básicos de un sistema de procesamiento digital de señales. La señal analógica es primero convertida a una señal de tiempo discreto mediante un circuito de muestreo y retención. La salida de este circuito es entonces aplicada a un circuito convertidor análogo-digital (ADC), en el cual la señal analógica muestreada es convertida a señal codificada digitalmente. El siguiente paso es aplicar la señal digital obtenida a un sistema de procesamiento de señal (PC, uC, DSP, uP) en donde un algoritmo es realizado. Dependiendo de la aplicación, algunas veces es necesario que la salida del procesador digital pueda ser usada directamente en forma digital pero en otras ocasiones se requiere volver a convertir la señal digital a una señal analógica, lo anterior se logra mediante un circuito convertidor digital-análogo (DAC).

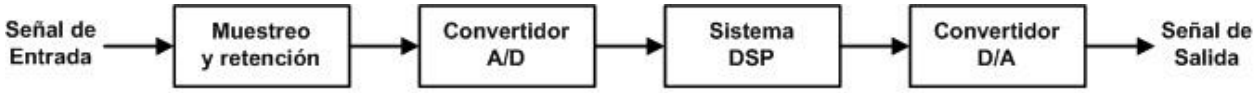


Figura 2.6 Sistema de procesamiento digital de señales.

En nuestro caso solo manejamos el sistema de procesamiento hasta el DSP, debido a que los datos enviados de éste hacia la PC será mediante el bus USB y son en formato digital, por lo que no contamos con el convertidor DAC.

2.6.1 Muestreo

El proceso de convertir una señal continua en tiempo a una señal discreta en tiempo es llamado *muestreo*. Este proceso consiste en tomar muestras de la señal continua cada cierto tiempo a intervalos constantes [28], en forma más precisa, este proceso puede ser considerado como la multiplicación de la señal continua en tiempo *g(t)* con un tren de pulsos unitarios *p(t)* y obteniéndose así la señal muestreada $g^{\#}(t)$, según fórmula 2.1, ver figura 2.7.

$$g^{\#}(t) = g(t)p(t) = \sum_{n=-\infty}^{+\infty} g(nT)\delta(t - nT) \qquad (2.1)$$

Debido a que los pulsos unitarios son solamente unos o ceros, la multiplicación puede ser considerada como una operación de switcheo.

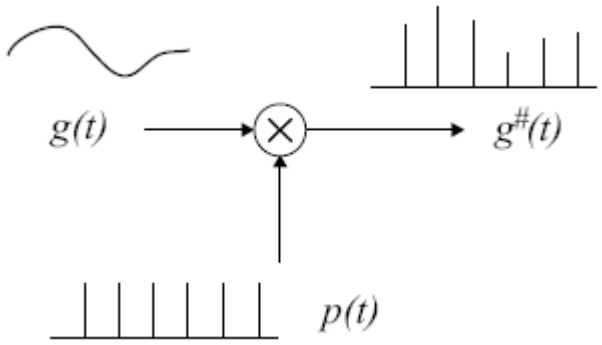


Figura 2.7 Muestreo visto como un proceso de multiplicación.

El periodo de tiempo *T* entre los pulsos unitarios del tren de pulsos es llamado *periodo de muestreo*. Este periodo *T* está relacionado a la *taza de muestreo* o *frecuencia de muestreo* f_s tal que:

$$f_s = \frac{w_s}{2\pi} = \frac{1}{T} \tag{2.2}$$

En una señal continua conocemos el valor de ésta en cualquier instante de tiempo, pero en una señal discreta en el tiempo (señal muestreada) solamente conocemos el valor de ésta en puntos específicos del tiempo. Para llevar a cabo correctamente el proceso de muestreo se utiliza el teorema de Nyquist, el Dr. Nyquist propuso el siguiente teorema de muestreo: “Una señal muestreada contiene toda la información sin ninguna distorsión, cuando la frecuencia de muestreo *(f_s)* es dos veces mayor a la frecuencia mayor *(f_{max})* de la señal muestreada”.

Este teorema demuestra que la reconstrucción exacta de una señal periódica continua a partir de sus muestras es matemáticamente posible si la señal está limitada en banda y la tasa de muestreo es superior al doble de su ancho de banda. Por el contrario si no se cumple con este criterio, existirán frecuencias cuyo muestreo coincide con otras, lo cual nos lleva a un problema llamado aliasing.

En este trabajo se tiene considerado frecuencias de muestreo de 200Hz hasta 1000Hz, para la señales bioeléctricas del cerebro; tal y como se menciona en el capítulo 2, estos

biopotenciales eléctricos van desde 0.5 Hz hasta 100 Hz, con lo cual se cumple con el teorema de muestreo.

Al llevar a cabo un estudio en el dominio de la frecuencia del proceso de muestreo se observa que éste crea un número infinito de copias del espectro de la señal original *g(t)*, en donde cada copia está desfasada por múltiplos de la frecuencia de muestreo w_s tal y como se puede apreciar en la figura 2.8. Por lo tanto existen dos situaciones al realizar un muestreo de señal, si $f_{max} < f_s/2$ las copias del espectro no estarán traslapadas, figura 3.4(a). Si por el contrario $f_{max} \geq f_s/2k$ entonces las copias del espectro estarán traslapadas, figura 2.8 (b).

Para eliminar el efecto alias en casos prácticos, el dispositivo de muestreo es siempre precedido por un filtro pasa bajos (anti-alias), el cual tiene el propósito de disminuir el ancho de banda de la señal continua dentro de las frecuencias de interés.

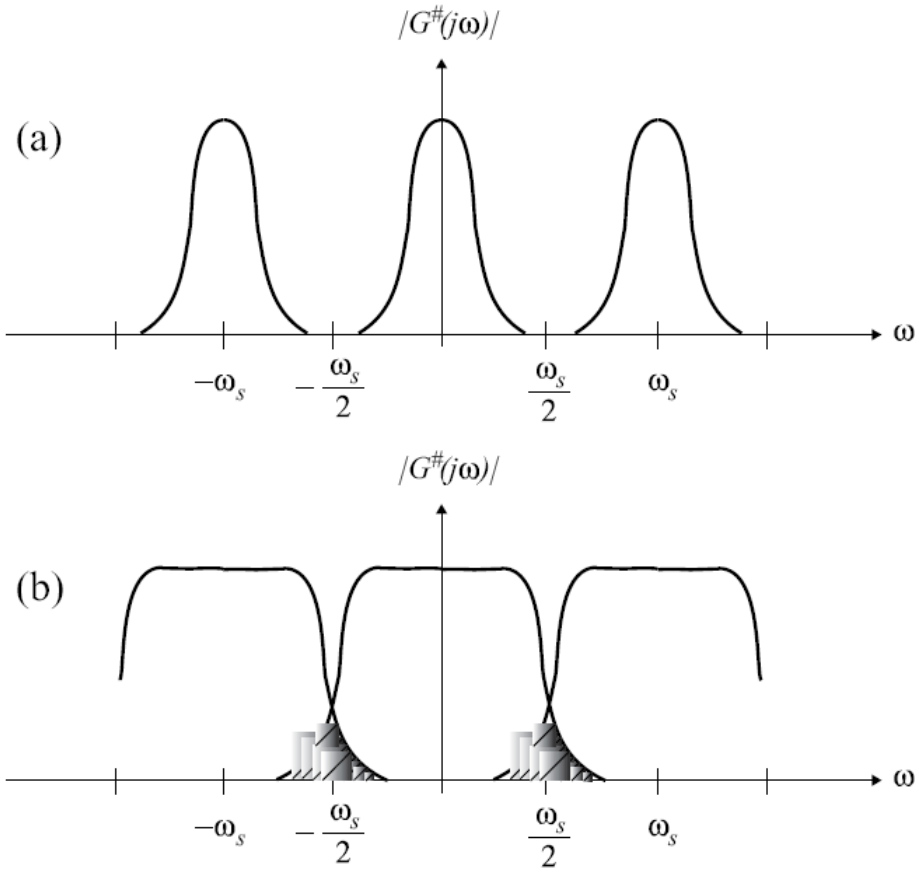


Figura 2.8 Parte del espectro de una señal muestreada.

Debido a que prácticamente es imposible construir filtros perfectos, existe el riesgo de que componentes de muy alta frecuencia se introduzcan en la señal muestreada, causando la distorsión llamada "*alias*", otra causa del alias podría ocurrir debido a que señales de muy alta frecuencia se introducen en el camino después de realizar el filtro anti-alias.

2.6.2 Cuantificación

El proceso de muestreo descrito arriba es el proceso encargado de convertir una señal continua en el tiempo a una señal discreta en el tiempo, mientras que el proceso de cuantificación es el encargado de realizar la conversión de una señal la cual es continua en amplitud a una señal discreta en amplitud, ver figura 2.9.

La mayoría de los sistemas hoy en día, utilizan el código binario, donde el número de cuantificación de los intervalos *N*, o resolución es:

$$\mathrm{N} = 2^{\mathrm{n}} \tag{2.3}$$

donde n es la longitud del dato binario. Por ejemplo en nuestro caso los biopotenciales serán cuantificados con n=12 bits con lo cual en este sistema obtendremos una resolución de *N*=4096, es decir los valores continuos de las señales de entrada al sistema de adquisición de datos, serán divididos sólo en 4096 valores distintos. Obviamente entre más bandas se utilicen para representar los biopotenciales la resolución del sistema será mejor y por lo tanto se tendrá una representación más precisa de la señal original.

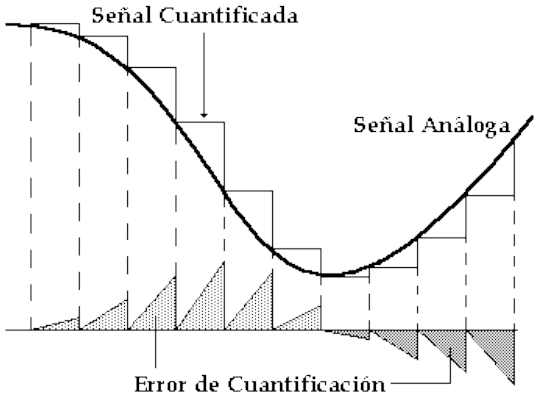


Figura 2.9 Cuantificación de señal análoga.

2.6.3 Filtro anti-alias

Para prevenir el fenómeno de alias en un sistema de adquisición de datos, es ampliamente utilizado el uso de filtros analógicos pasa bajos justo antes de los convertidores análogo a digital (ADC), para limitar el ancho de banda de la señal analógica que será muestreada [29]. Estos filtros son llamados Filtro anti-alias.

En el presente trabajo se tiene filtros anti-alias, para cada canal, con una frecuencia de corte de 100 Hz a la salida de cada señal del amplificador del EEG para así cumplir con esta etapa. Este filtrado se conserva de la versión de adquisición de datos con la tarjeta comercial de NI. El filtro implementado en la etapa de acondicionamiento del EEG, es un filtro pasa bajos RC.

2.6.4 Arquitecturas de convertidores análogo-digital más usuales.

El convertidor análogo-digital, ADC, convierte una señal analógica a una palabra digital. Los convertidores convencionales trabajan muestreando la señal analógica variante en el tiempo a una frecuencia suficiente para alcanzar las componentes de más alta frecuencia de dicha señal, según el teorema de muestreo. Por lo anterior la frecuencia de muestreo deberá ser el principal requerimiento para escoger una arquitectura de convertidor apropiada. Ciertas arquitecturas explotan el paralelismo para lograr altas velocidades de operación en el orden de 100’s de MHz, y otros pueden ser usados para alta precisión, 16 ó 24 bits de resolución para señales con frecuencias máximas en el orden de 10’s de KHz. Existen diversos tipos de arquitecturas para un ADC, como son, *Flash, de aproximaciones sucesivas y Sigma-Delta*.

El ADC utilizado para este trabajo, TLV2556, pertenece a la categoría de Sigma-Delta.

2.7 FILTROS DIGITALES

En muchas aplicaciones del procesamiento de señales es necesario diseñar dispositivos o algoritmos que realicen operaciones sobre las señales y que los englobaremos bajo la denominación genérica de Sistemas.

Un filtro digital es un sistema de tiempo discreto que realiza alguna transformación sobre una señal de entrada digital *x(n)*, generando una secuencia de salida *y(n)*, tal y como se aprecia en la figura 2.10.

$$y(n) = T[x(n)]$$

Figura 2.10 Diagrama de bloque de un filtro digital**.**

El comportamiento de un filtro es usualmente descrito en términos de la relación de entrada-salida. Estos usualmente son evaluados introduciendo al filtro diferentes entradas y evaluando cual es la respuesta del sistema a cada una de las entradas. Si la entrada es la secuencia de impulsos δ*(n)*, la respuesta al impulso, juega un papel muy importante para describir la característica del filtro, en la ecuación 2.4 se plantea matemáticamente un filtro digital genérico.

$$x(n) = \sum_{k=-\infty,\infty} x(k) \cdot \delta(n-k) \tag{2.4}$$

Siendo *h(n-k)* la respuesta a δ*(n-k), la ecuación queda:*

$$y(n) = \sum_{k=-\infty,\infty} x(k) \cdot h(n-k) \tag{2.5}$$

Como se puede apreciar, la ecuación 2.5, está basada en la convolución y la cual define la propiedad del filtro. Un filtro debe ser estable pero pudiera ser o no casual.

Un filtro es estable si y solo si,

$$\sum_{k=-\infty,\infty} |h(k)| < \infty \tag{2.6}$$

Casualidad significa que el filtro no responderá a una entrada antes de que ésta sea aplicada, un filtro es casual si y solo si *h (k)=0* para *k<0*.

Debido a la utilización de la convolución como una pieza clave en la implementación de un filtro digital en la sección 3.3.1 es explicada.

Al igual que en los filtros analógicos, existen una amplia variedad de clasificaciones para estos filtros. De acuerdo con la parte del espectro que dejan pasar y que atenúan, existen los siguientes: Pasa alto, pasa bajo, pasa banda, rechaza banda. De acuerdo con su orden, primer orden, segundo, etc. De acuerdo al tipo de respuesta ante una entrada unitaria, se tiene, FIR e IIR.

Actualmente los filtros digitales son utilizados con mayor frecuencia en cada equipo electrónico por sencillo que parezca, el uso de filtros digitales ha sido propuesto en este trabajo para filtrar las señales bioeléctricas del EEG en el procesador de señales digitales (DSP) y no en la PC como hasta ahora se ha estado realizando, el filtro utilizado para la aplicación de EEG es un filtro FIR pasa bandas en conjunto con un rechaza banda a 60 Hz y éste también puede ser aplicado para un ECG.

2.7.1 Convolución

La convolución es una manera matemática de combinar dos señales para formar una tercera. Es importante mencionar que esta técnica es la más importante en el Procesamiento de Señales Digitales [30], [31]. La convolución es importante porque relaciona las tres señales de interés: señal de entrada, señal de salida y la respuesta al impulso.

La respuesta al impulso es prácticamente la respuesta de un sistema lineal que se desea analizar al cual se le aplica a la entrada una señal de impulso (función delta), frecuentemente normalizada llamada impulso unitario, con esta prueba se puede verificar si dos sistemas son diferentes debido a que cada uno de estos sistemas tendrán una respuesta al impulso distinta, ver figura 2.12.

La función impulso es más un concepto matemático que una función, que se define de la siguiente manera:

$$\delta(t) = \begin{cases} \infty, & t = 0 \\ 0, & t \neq 0 \end{cases} \tag{2.7}$$

$$\int_{-\infty}^{+\infty} \delta(t)dt = 1 \tag{2.8}$$

La función es cero para cualquier valor de *t*, excepto en *t* = 0, donde el valor de la función es infinito, en la figura 2.11 se puede ver la representación gráfica de esta función.

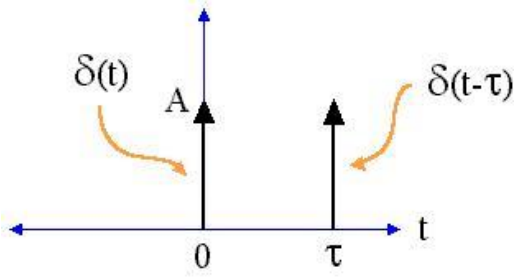


Figura 2.11 Función Delta ó impulso.

Conociendo la respuesta al impulso de un sistema, inmediatamente se sabrá cómo reaccionará el sistema a cualquier impulso.

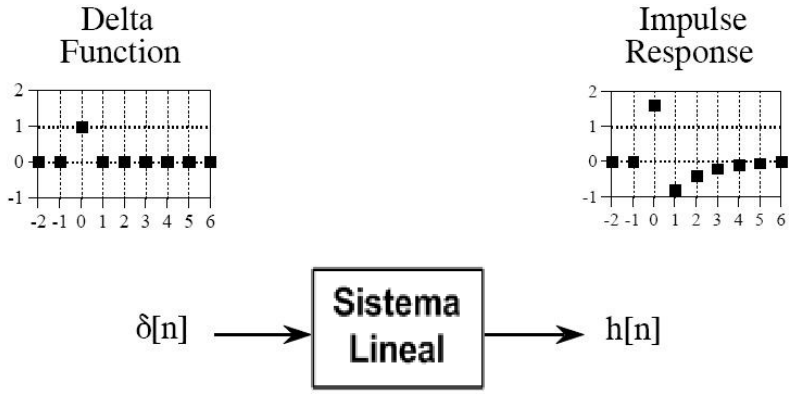
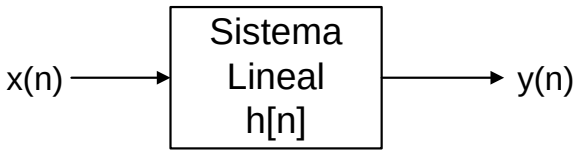


Figura 2.12 Respuesta al impulso.

Por lo anterior, al realizar la convolución de la señal de entrada con la respuesta al impulso del sistema, podemos conocer la señal de salida, figura 2.13. Usualmente la convolución es representada por el símbolo *, y ésta es utilizada ampliamente para la realización de filtros digitales, ver figura 2.14. En el presente trabajo estamos usando una convolución en el DSP precisamente para realizar el filtrado digital.



$$x[n] * h[n] = y[n]$$

Figura 2.13 Convolución.

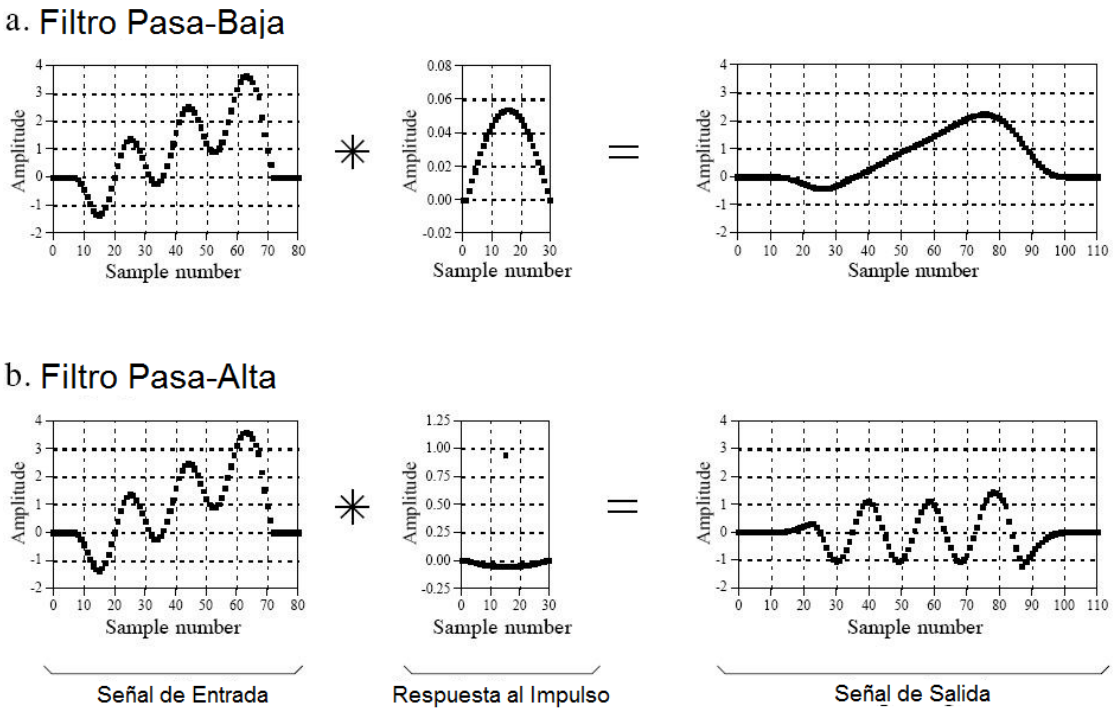


Figura 2.14 Aplicación de la convolución en filtros digitales.

En la mayoría de las aplicaciones de DSP, la señal de entrada es de cientos, miles o inclusive millones de muestras, la respuesta al impulso usualmente es mucho más corta, matemáticamente la convolución no restringe la longitud de estas señales, sin embargo, si se especifica la longitud de la señal de salida. La longitud de la señal de salida es igual a la suma de la longitud de la señal de entrada más la longitud de la respuesta al impulso menos uno.

Si trasladamos a una fórmula el funcionamiento de la convolución en el ámbito digital, tenemos lo siguiente:

$$y[i] = \sum_{j=0}^{M-1} h[j]x[i-j] \qquad (2.9)$$

En donde:

- *y[i]* → Señal de salida
- *i* → i-esima muestra de la señal de salida
- *M* → La cantidad de datos de la respuesta al impulso
- *h[j]* → Respuesta al impulso
- *j* → j-esima muestra de la respuesta al impulso

A continuación se explicarán tanto los filtros al impulso finito (FIR) y filtros al impulso infinito (IIR), como ya se mencionó en el presente trabajo se utiliza solamente filtro FIR, para la implementación de un filtro pasa banda y uno rechaza banda de 60Hz.

2.7.2 Respuesta al impulso finito (FIR)

Los filtros digitales de Respuesta Finita al Impulso o filtros FIR por sus siglas en inglés Finite Impulse Response, se trata de un tipo de filtros en el que, como su nombre lo indica, si la entrada es una señal impulso la salida tendrá un número finito de términos no nulos. La estructura de señal a la salida del filtro se basa solamente en la combinación lineal de las entradas actuales y anteriores, esto es:

$$y[n] = \sum_{k=0}^{N-1} b_k x[n-k] = \sum_{k=0}^{N-1} h[k]x[n-k] \;\; con\; h[k] = \{h_0\; h_1 \ldots\; h_{N-1}\} \qquad (2.10)$$

donde N es el orden del filtro, que también coincide con el número de términos no nulos y con el número de coeficientes b_k del filtro. Observe que la expresión de la ecuación 2.10 corresponde a la convolución de la señal de entrada *y[i]* con la respuesta al impulso del filtro FIR *h[n]*, ecuación 2.9.

Aplicando la transformada Z a la respuesta al impulso del filtro FIR *h[n]*, se tiene:

$$H(z) = \sum_{k=0}^{N-1} h_k z^{-k} \;=\; h_0 + h_1 z^{-1} + \cdots + h_{N-1} z^{-(N-1)} \qquad (2.11)$$

En la figura 2.15, se muestra el diagrama en bloques de la estructura básica del filtro FIR, para una cantidad de 12 coeficientes.

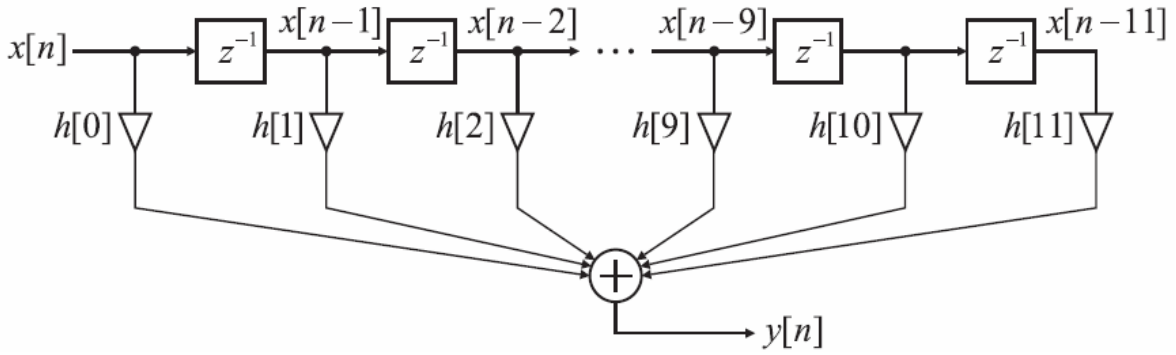


Figura 2.15 Representación en diagrama a bloques del filtro FIR con 12 coeficientes.

Ya que un filtro FIR ideal es aquel sistema discreto cuya salida es una versión escalada, a través de un factor A_o, y desfasada de la entrada, por *D* muestras, es decir:

$$y[n] = A_0 x[n - D] \tag{2.12}$$

El filtro no distorsiona la excitación x[n] al permitir el paso de sus componentes frecuenciales en el rango de interés. Así pues se tiene que la función de transferencia es:

$$H(z) = \frac{Y(z)}{X(z)} = A_0 z^{-D} \tag{2.13}$$

Siendo la respuesta en frecuencia o la transformada de Fourier a tiempo discreto del filtro FIR igual a:

$$H\left(e^{j\Omega}\right) = H(\Omega) = A_0 e^{-j\Omega D} \tag{2.14}$$

De la ecuación 2.14 se observa que la magnitud $H(e^{j\Omega})$ es constante en el rango de frecuencias para el cual fue diseñado el filtro, y la fase varia linealmente con la frecuencia.

En la práctica, un filtro implementado físicamente tiene bandas de paso, transición y rechazo y no sólo frecuencias de corte, tal como se muestra en la figura 2.16, correspondiente a un filtro pasa bajos.

El siguiente paso sería obtener los coeficientes del filtro FIR cuya respuesta cumpla las especificaciones. Asimismo, se observa que el filtro FIR es estable ya que su función de transferencia discreta no tiene polos fuera del círculo unitario, ver ecuación 2.10.

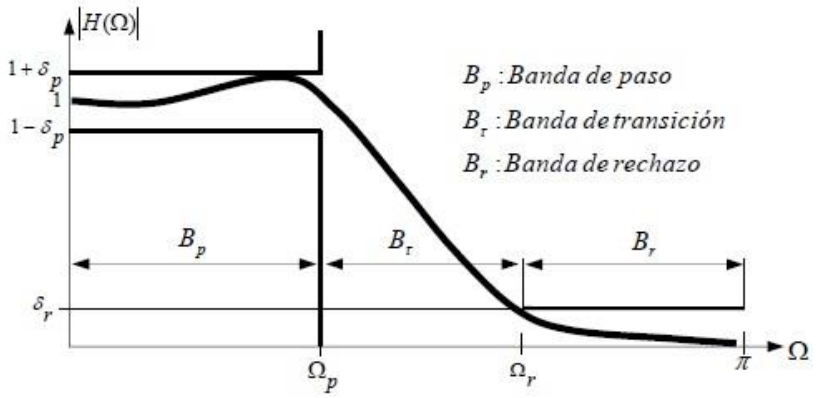


Figura 2.16 Parámetros de un filtro digital pasa bajas.

Para diseñar un filtro FIR es necesario tener en cuenta que la cantidad de coeficientes o duración de la respuesta al impulso del filtro es siempre finita, a diferencia de la de la respuesta al impulso de su respectivo filtro ideal. Por lo que la respuesta al impulso del filtro FIR exhibirá cierto truncamiento implícito en comparación con la respuesta al impulso del filtro ideal. Este truncamiento se manifiesta en la respuesta en frecuencia del filtro, en la cual se producen ondulaciones antes y después de cualquier discontinuidad. Es por ello que comúnmente se utilizan ventanas a través de un número finito de secuencias de *w[n]*, con el fin de disminuir principalmente los rizos de la banda de rechazo en la respuesta en frecuencia del filtro.

En la figura 2.17, se muestra una gráfica de tal situación y en la tabla 2.3 se muestran las ventanas típicas utilizadas [28].

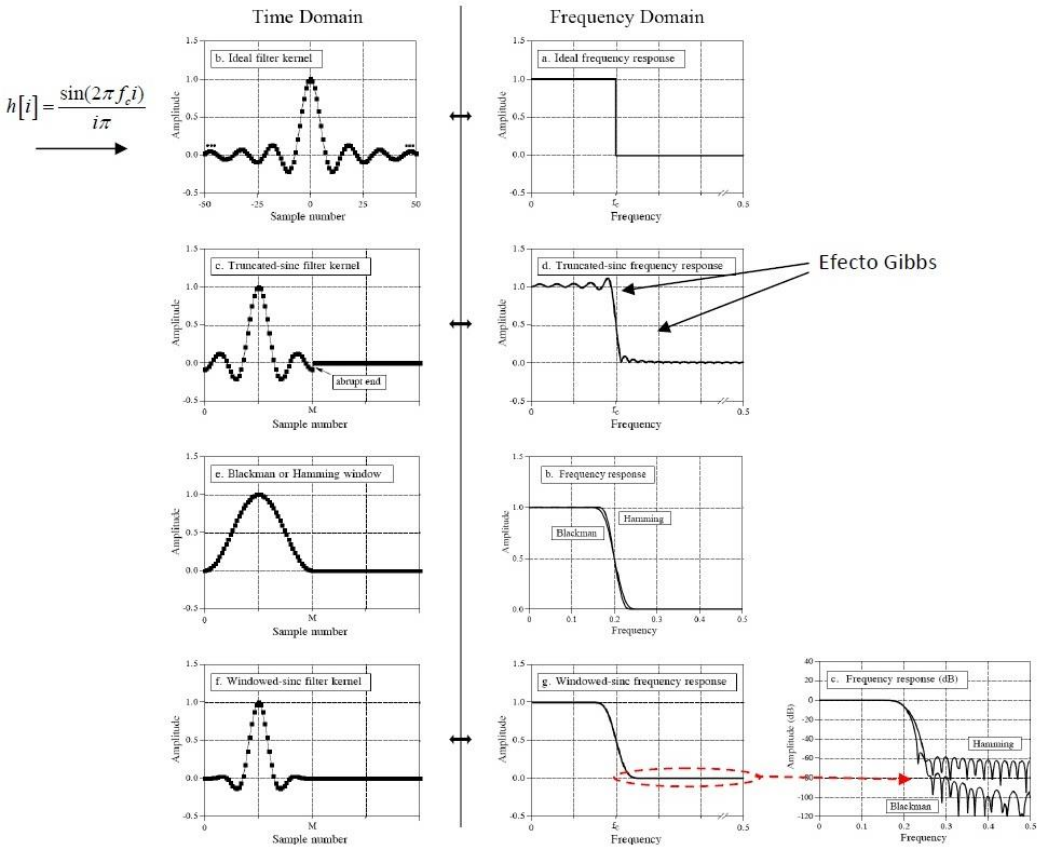


Figura 2.17 Concepto del diseño de filtros digitales utilizando ventanas.

Tipo de Ventana	Expresión Matemática
Rectangular	$w[n] = 1 \ para \ 0 \leq n \leq N-1$ $w[n] = 0 \ para \ cualquier \ otro \ valor$
Barlett ó Triangular	$w[n] = \frac{2n}{N-1} \ para \ 0 \leq n \leq \frac{N-1}{2}$ $w[n] = 2 - \frac{2n}{N-1} \ para \ \frac{N-2}{2} \leq n \leq N-1$ $w[n] = 0 \ para \ cualquier \ otro \ valor$
Hann	$w[n] = \frac{1}{2}\left[1 - \cos\left(\frac{2\pi n}{n-1}\right)\right] \ para \ 0 \leq n \leq N-1$ $w[n] = 0 \ para \ cualquier \ otro \ valor$
Hamming	$w[n] = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right) \ para \ 0 \leq n \leq N-1$ $w[n] = 0 \ para \ cualquier \ otro \ valor$
Blackman	$w[n] = 0.42 - 0.5\cos\left(\frac{2\pi n}{N-1}\right) + 0.08\cos\left(\frac{4\pi n}{N-1}\right) \ para \ 0 \leq n \leq N-1$ $w[n] = 0 \ para \ cualquier \ otro \ valor$
Kaiser	$w[n] = \frac{I_0\left[\beta\sqrt{1-\left(\frac{2n}{N-1}\right)^2}\right]}{I_0[\beta]} \ para \ 0 \leq n \leq N-1$ $w[n] = 0 \ para \ cualquier \ otro \ valor \ de \ n$ $siendo \ I_0 \ la \ función \ de \ Bessel \ de \ primer \ tipo \ de \ orden \ cero \ y \ \beta \ determina \ la \ forma \ de \ la \ ventana$

Tabla 2.3 Ventanas utilizadas en las operaciones de filtrado digital.

Las características principales de los filtros FIR es que no cuentan con retroalimentación, por lo cual cuenta con mayores localidades de memoria. Por otro lado, estos filtros son siempre estables, por lo general son de fase lineal y son más fácil de implementar que un filtro IIR.

A la hora de realizar el diseño de los filtros, se busca tener cierto comportamiento ya sea en el dominio del tiempo como en el dominio de la frecuencia. Dicho comportamiento comienza por una “cuestión estética”, que se traduce en la estabilidad, distorsión y ubicación espacial de los ceros del filtro.

2.7.3 Respuesta al impulso infinito (IIR)

La ecuación básica para el filtro IIR es la misma que para cualquier proceso lineal general mostrado en la ecuación 2.15.

$$y(k) = \sum_{n=1}^{L_N} b(n)x(k-n) - \sum_{n=1}^{L_D} a(n)y(k-n) \qquad (2.15)$$

donde *b(n)* corresponde a los coeficientes del numerador, el cual también es utilizado en los filtros FIR, *a(n)* es los coeficientes del denominador, *x(n)* es la entrada y *y(n)* la salida. Los coeficientes correspondientes al numerador, *b(n)*, operan únicamente sobre los valores de entrada, *x(n)*, mientras que los coeficientes del denominador, *a(n)*, operan sobre valores anteriores de la salida, *y(n)*, por lo que algunas veces son llamados como coeficientes recursivos.

La mayor desventaja de estos filtros es que tienen características de fase no lineal.

El diseño de filtros IIR no es tan sencillo como el de filtros FIR. Ya que los filtros IIR tienen funciones de transferencia que son las mismas de un proceso lineal general, las cuales contienen tanto polos y ceros, muchos de los conceptos del diseño de filtros analógicos pueden ser usados con estos filtros. Determinar el número de polos requeridos en un filtro IIR dando una característica de atenuación deseada es un proceso sencillo.

Existen dos métodos generales para el diseño de filtros digital de respuesta al impulso infinita, IIR. El más común consiste en diseñar un filtro analógico IIR para luego mapearlo a su equivalente digital.

El segundo método emplea un procedimiento de diseño algorítmico, que por lo general requiere del uso de computadoras para resolver un conjunto de ecuaciones lineales y no lineales. Este método es utilizado para resolver filtros con respuesta en frecuencia arbitraria, no caracterizados por las respuestas en frecuencia de filtros analógicos existentes.

Para el diseño de un filtro digital desde su prototipo analógico requiere transformar $h_a(t)$ en $h_a(n)$ o lo que es lo mismo, $H_a(s)$ en $H_a(z)$. El mapeo del plano s en el plano z puede ser escrito como.

$$H_a(z) = H_a(s)|_{s=m(z)} \quad (2.16)$$

donde *m(z)* es la función de mapeo. Para que la transformación produzca un filtro digital aceptable el mapeo *m(z)* debe cumplir con las siguientes propiedades:

1. Mapeo del eje *j*Ω al círculo unidad $|z| = 1$. Este mapeo debe ser uno a uno sobre el círculo unidad en función de preservar la característica de respuesta en frecuencia del filtro analógico.
2. Los puntos del semiplano izquierdo deben ser mapeados en puntos dentro del círculo unidad para preservar la estabilidad de los filtros analógicos.
3. El mapeo *m(z)* debe ser una función racional de z para que $H_a(s)$ sea mapeada a una función racional en z.

Para el desarrollo actual se utiliza en cuanto al procesamiento de la señal un filtro pasabanda de 0.5Hz a 100Hz con un rechazabanda de 60Hz, estos dos últimos son tipo FIR y son aplicados con la convolución.

CAPÍTULO 3 – CONCEPTOS TECNOLÓGICOS

En el presente capítulo se expondrán los conceptos tecnológicos utilizados para este desarrollo, primeramente se mencionará la teoría sobre los procesadores digitales de señales (DSP) considerando su arquitectura, memoria y periféricos.

De manera adicional, se abordará otra parte fundamental en el desarrollo presentado en este trabajo, el puerto universal serial, mejor conocido como USB, se dará una introducción a este puerto y se mencionarán los diferentes conceptos importantes para un desarrollo utilizando este protocolo de comunicación.

3.1 PROCESADOR DIGITAL DE SEÑALES (DSP)

Un procesador digital de señales (DSP) se ha utilizado principalmente en aplicaciones donde se requiera del uso de operaciones matemáticas para el análisis de señales digitales en diversas áreas de la ingeniería, incluyendo la industria biomédica para el desarrollo de equipos médicos [32].

Actualmente, los DSP's se han convertido en elementos muy comunes en el diseño electrónico, sustituyendo en algunas aplicaciones a los microprocesadores de propósito general y microcontroladores, debido a sus ventajas de procesamiento, consumo de energía y el tiempo de desarrollo.

El uso de los DSP's actualmente es determinante para la industria electrónica, por ejemplo, los sistemas de visión incorporados a los automóviles, permiten que estos puedan manejarse de manera autónoma, la telemedicina evoluciona constantemente en el procesamiento del análisis en tiempo real evitando latencia de la información, entre muchos otros avances tecnológicos.

El enfoque de los próximos 20 años es profundo, de manera que aumentarán los desafíos. Para traer portabilidad, conectividad e inteligencia a cada dispositivo electrónico, las compañías de semiconductores necesitarán ofrecer soluciones completas, que involucren hardware, software y herramientas de diseño en el procesamiento de señales. Con la especialización tecnológica y colaboración de la industria, los desarrollos tendrán que tener mejores rendimientos, precio y consumo de energía [33]. En la figura 3.1 se puede observar la cronología de los DSP's a partir de que éstos se desarrollaron.

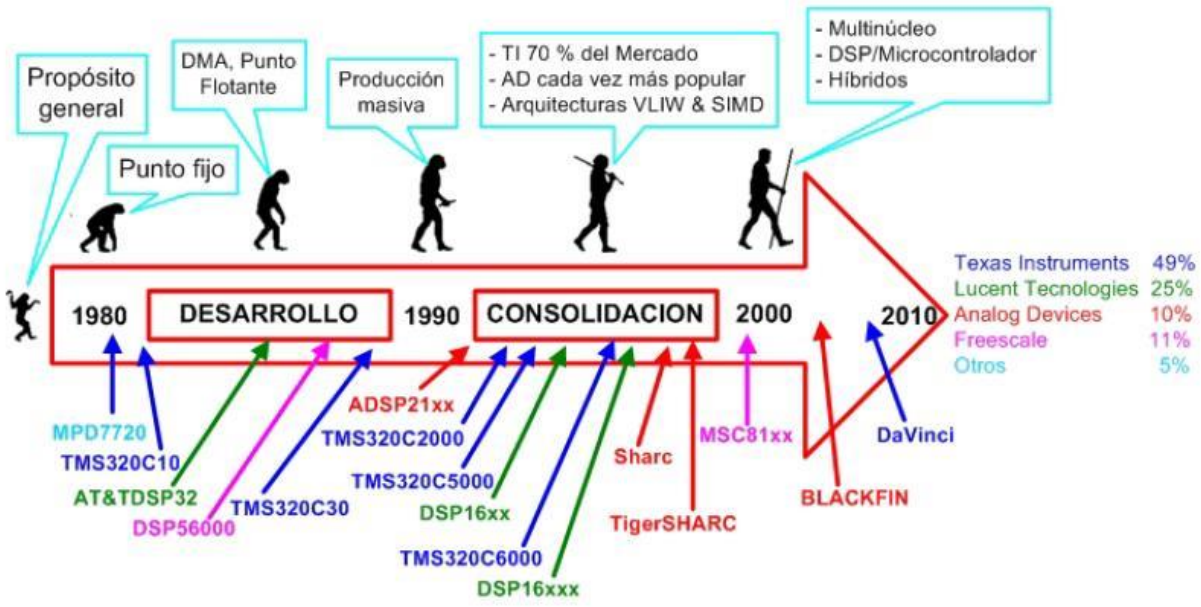


Figura 3.1 Evolución de la tecnología de los DSP.

3.1.1 Características generales

La estructura general que describe a un sistema para procesamiento digital de señales responde al diagrama de bloques que se muestra en la figura 3.2.

La señal analógica se obtiene a través de un sensor o transductor que transforma una magnitud física en una señal eléctrica. Un convertidor analógico-digital convierte la señal analógica en una secuencia numérica. Estas muestras llegan a un elemento procesador en el cual se ejecuta algún algoritmo de procesamiento digital. La salida de este procesador se introduce a un convertidor digital-analógico para nuevamente obtener una señal analógica, la que a su vez se puede transformar en una magnitud física por medio de un actuador.

Para este proyecto no se utiliza la parte del convertidor D/A ni el actuador, ya que en el trabajo aquí desarrollado una vez que se realiza el procesamiento de la señal, ésta se envía a la PC vía el puerto de comunicación USB, para el despliegue y almacenamiento de las señales del EEG.

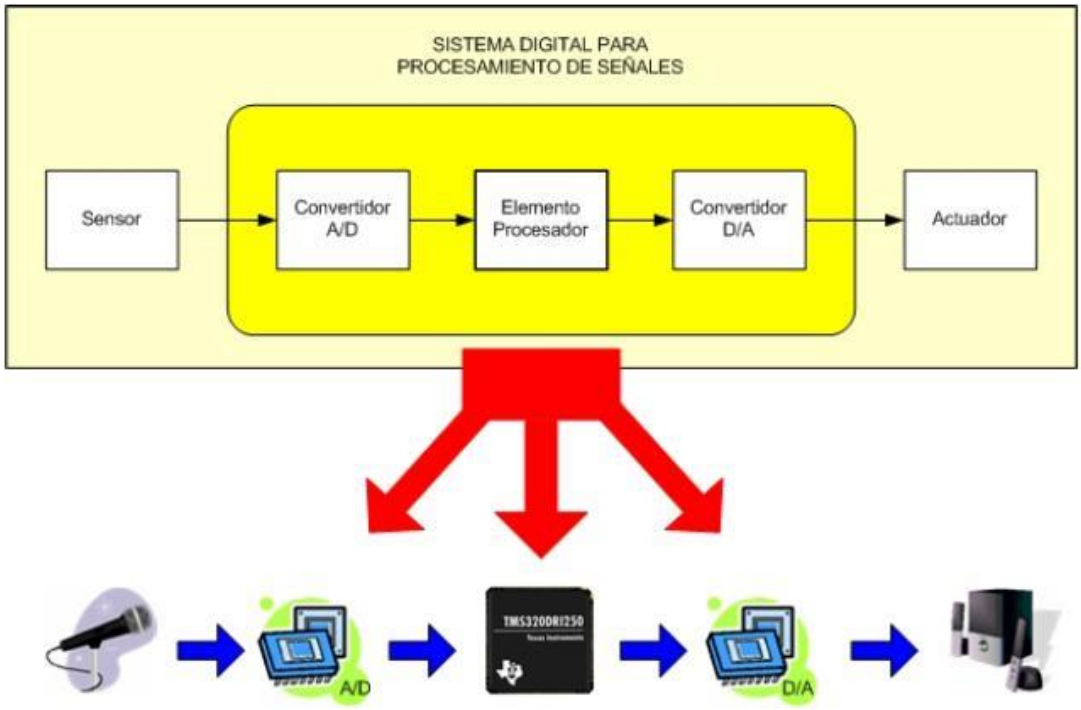


Figura 3.2 Configuración típica de un sistema de procesamiento digital.

Un aspecto sumamente importante es que las características del procesador a utilizar estarán impuestas por los requerimientos de las aplicaciones en que se han de utilizar, y en la naturaleza de las operaciones de la señal que es necesario realizar. La operación básica es el producto acumulativo de dos secuencias, a la cual se le denomina operación MAC (multiply and acumulate), ésta se implementa en aplicaciones como: filtrado, análisis espectral, correlaciones, etc. Los factores del producto pueden ser, dependiendo del caso, muestras de una señal, coeficientes de un filtro o constantes precalculadas, como tablas de senos y cosenos.

En resumen, si se consideran las condiciones que determina una aplicación, las características generales que debe reunir el elemento procesador son:

- Deben ser dispositivos con una arquitectura que permita procesar las muestras de entrada a una gran velocidad, ya que en ocasiones son en tiempo real.
- Los algoritmos de procesamiento digital de señales tienen una elevada carga computacional en los que predomina la operación MAC, por lo que este elemento procesador debe tener una gran capacidad de cálculo.

- Dado el gran volumen de datos a procesar deben permitir un manejo accesible de los mismos.
- Tendrán que ser programables para hacer posible la implementación de distintos tipos de algoritmos de procesamiento digital de señales.

3.1.1.1 ALGORITMOS

Los sistemas de procesamiento digital de señales, frecuentemente se caracterizan por la utilización de un algoritmo que especifica las operaciones aritméticas a realizar con los datos. La implementación del algoritmo se puede realizar mediante software en un microprocesador, un procesador, o mediante hardware a la medida.

Dentro de la implantación del algoritmo en software se encuentran dos alternativas. La primera de ellas, utiliza microprocesadores de propósito general de altas prestaciones, como por ejemplo el Pentium de Intel, Power Pc 601 de Motorola e IBM, etc. Los sistemas que se basan en este tipo de procesadores son muy complejos porque usan una serie de periféricos y elementos externos para su implementación. De ahí que se conozcan como procesadores de propósito general. Además, estas altas prestaciones no siempre se traducen en una elevada potencia de cálculo o una rápida ejecución de operaciones tipo MAC, debido a las restricciones en la ejecución de operaciones en punto flotante.

La otra alternativa para la implementación del algoritmo en software son los procesadores digitales de señales (DSP's), los cuales tienen una arquitectura específicamente diseñada para estas aplicaciones (arquitectura Harvard). Son dispositivos con prestaciones superiores y de menor costo que los de propósito general. Además, disponen de herramientas de desarrollo flexibles que permiten acelerar el proceso de diseño.

En lo que respecta a la implementación de algoritmos en hardware, ésta se realiza mediante circuitos integrados (CI) diseñados a la medida, integrando en el dispositivo sólo aquellas funciones necesarias para la aplicación concreta a que van destinados. Este hardware puede tomar múltiples formas. Una de ellas son los dispositivos estándar para aplicaciones específicas (ASSP, por sus siglas en inglés) [37]. Como su nombre indica, se trata de un circuito integrado que implementa un algoritmo concreto. Una

ventaja es que pueden operar con señales de alta frecuencia, pero al mismo tiempo se dificulta el diseño del dispositivo por lo que los algoritmos implementados deben de ser sencillos, de una estructura regular y con un número limitado de bits de operación. Un ejemplo de los ASSP son los filtros digitales programables, los cuales integran un conjunto de multiplicadores en hardware que operan en paralelo para implementar un filtro FIR, por ejemplo.

Finalmente, los DSP's reúnen una serie de elementos que los hacen diferentes del resto de los procesadores. Estas características se manifiestan en:

- La arquitectura Harvard del CPU, que dispone de recursos que posibilitan las realizaciones de las operaciones MAC de forma rápida, y la utilización de modos de direccionamiento especiales para el manejo de buffers.
- Un juego de instrucciones optimizado para aplicaciones de tratamiento digital de señales, entre las que destacan aquellas que le permiten realizar un lazo de control eficiente.
- Una arquitectura de memoria que le permita obtener las instrucciones y datos a procesar a la velocidad que los demanda el CPU.
- La inclusión de un conjunto de periféricos en el mismo dispositivo que le permita comunicarse con el exterior.

3.1.1.2 VELOCIDAD DE RELOJ

Los sistemas electrónicos digitales se caracterizan por su velocidad de reloj. La velocidad de reloj se refiere a la velocidad a la cual el sistema ejecuta sus unidades de trabajo más básicas. Para los sistemas con DSP's, la relación de la velocidad del reloj y la velocidad de muestreo es una de las características más importantes usadas para determinar cómo será implementado el sistema. Es decir, esta relación determina parcialmente el monto de hardware necesario para implementar un algoritmo con una complejidad dada [34].

3.1.1.3 VELOCIDAD DE MUESTREO

Una característica clave de los sistemas con DSP's es su tasa de muestreo (sample rate). El muestreo es el proceso de convertir una señal en tiempo continuo a una señal

en tiempo discreto, en intervalos de muestreo también discretos. Las amplitudes de las señales en tiempo discreto se cuantifican en valores digitales con un ancho de palabra $N=2^n$ (donde *n* es el número de bits) dado. Un ADC realiza los procesos de muestreo y cuantificación de una señal como se observa en la figura 3.3.

f_M se puede replicar con exactitud con los valores muestreados si la tasa de muestreo, *fs*, es superior al doble de la frecuencia máxima de la señal de entrada. Es decir, $fs \geq 2 f_M$ [36].

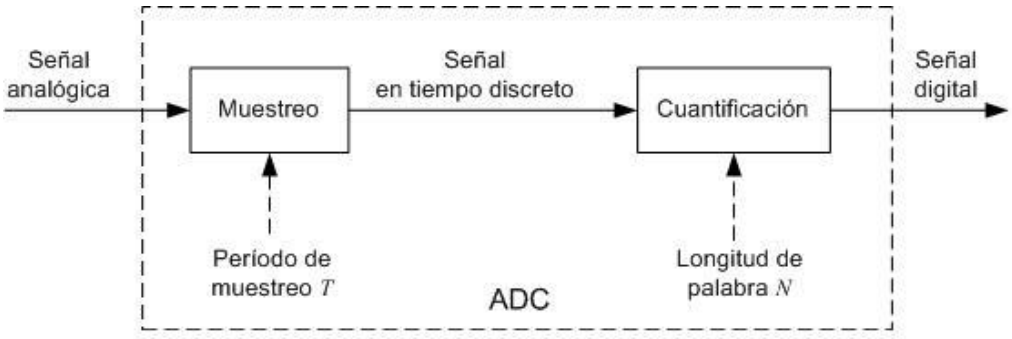


Figura 3.3 Proceso de conversión analógico-digital.

Si se utiliza una frecuencia menor a la establecida por el teorema de Nyquist, se produce una distorsión conocida como traslape espectral (aliasing). El traslape impide recuperar correctamente la señal cuando las muestras de ésta se obtienen a intervalos de tiempo demasiado largos. La forma de la onda recuperada presenta pendientes muy abruptas.

Aunque la mayoría de los DSP's incluyen entre sus periféricos algún ADC, en nuestro caso utilizaremos un ADC externo comunicado hacia el DSP por una interfaz serial.

3.1.1.4 FORMATOS DE DATOS

Otra característica importante que ayuda a determina la correcta selección de un DSP para una aplicación dada, es el tipo de formato y número de bits de los datos con que realiza los cálculos matemáticos. Con relación al tipo de formato de datos, los DSP's pueden operar con números en punto fijo, punto flotante o ambos.

Algunos DSP's sólo son capaces de operar con números enteros, números en punto fijo, salvo que se considere la existencia de un punto binario, mediante el cual se

escalan los valores enteros para que de esta forma obtener números fraccionarios. Este factor de escala es igual a 2^{-bp}, donde *bp* es la posición del punto binario.

El DSP realiza las operaciones de suma o multiplicación como si se tratara de números enteros, sin considerar este factor de escala. Es responsabilidad del programador interpretar la posición del punto binario. En la figura 3.4 se encuentran algunos ejemplos de la representación de números en aritmética de punto fijo.

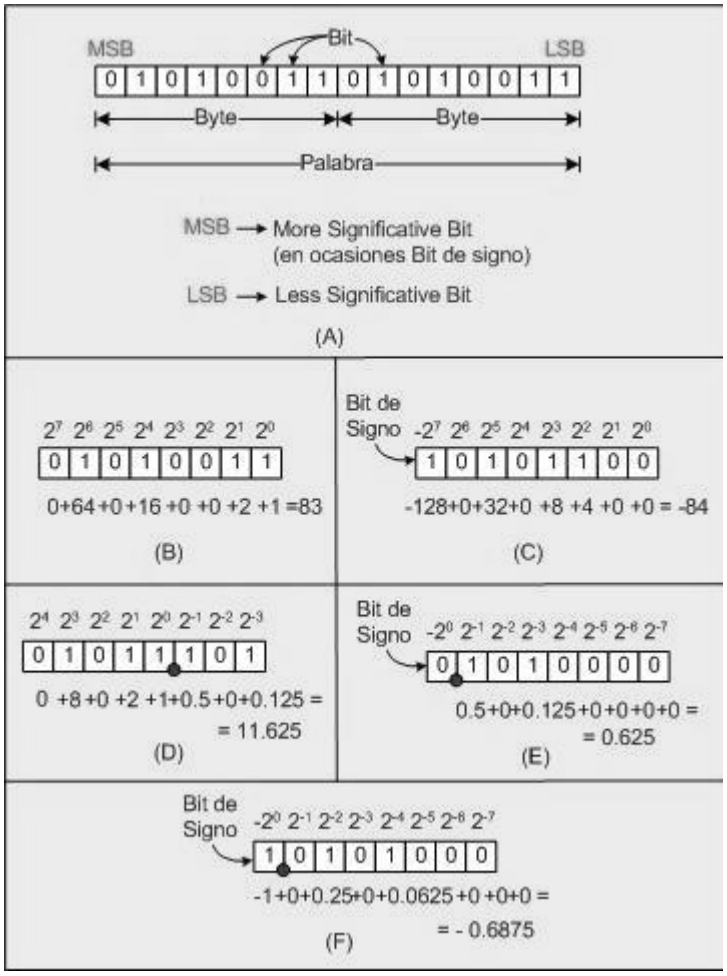


Figura 3.4 Representación de números en punto fijo. (A) Ancho de palabra. (B) Entero Positivo. (C) Entero Negativo, (D) Entero y fraccional, (E) Fraccional positivo. (F) Fraccional negativo.

Al desplazar el punto decimal a la izquierda, utilizando más bits para la parte fraccionaria, la precisión aumenta, pero disminuye el margen de valores de la representación. Puesto que el tamaño de la palabra de datos es fijo, la situación del punto binario será una situación de compromiso entre la precisión a obtener y el margen de valores a cubrir. El programador debe utilizar el mayor número de bits para

la parte fraccionaria (máxima precisión) que permita representar todo el intervalo de valores que toma una variable.

Si durante el procesamiento, un número en punto fijo aumenta demasiado para poder ser representado con el número de bits disponibles para la parte entera, el programador debe realizar un escalamiento descendiente del número mediante un desplazamiento a la derecha, perdiendo los bits de menor peso y por tanto disminuyendo la precisión. Si por el contrario el número en punto fijo disminuye demasiado, el número de bits utilizados en la parte fraccionaria puede ser insuficiente. El programador entonces deberá hacer un desplazamiento a la izquierda para aumentar la precisión.

En ambos casos el programador debe tomar en consideración como se ha ido desplazando el punto binario, restaurando todos los números de punto fraccionario a una misma escala en una etapa posterior. Esto convierte la programación en una tarea muy tediosa.

Existen algunas rutinas o librerías en punto fijo, desarrolladas por los propios fabricantes de DSP´s, que emulan las operaciones de punto flotante (lo cual permite manejar cómodamente los números fraccionarios) para el caso del DSP seleccionado para este trabajo, Texas Instruments cuenta con una librería denominada Q15, la cual nos permite manejar las operaciones de punto fijo como si fueran de punto flotante.

Otros procesadores disponen de un CPU capaz de operar directamente con números en punto flotante. La aritmética de punto flotante es un mecanismo más flexible que la de punto fijo, ya que con la primera los diseñadores tienen acceso a un intervalo de valores mucho más amplio y a una mejor precisión, ver la figura 3.5. Esto facilita la programación, ya que no es necesario preocuparse por el escalamiento. Un número de punto flotante se representa mediante una mantisa y un exponente, ver la ecuación

$$Valor = Mantisa\ x\ 2^{exponente} \tag{3.1}$$

La mantisa es un número fraccionario, mientras que el exponente determina la posición del punto binario. En estos procesadores, es el propio hardware del CPU el que realiza los escalamientos mencionados anteriormente, quedando reflejada la posición del punto binario en el exponente. Esto facilita enormemente la programación.

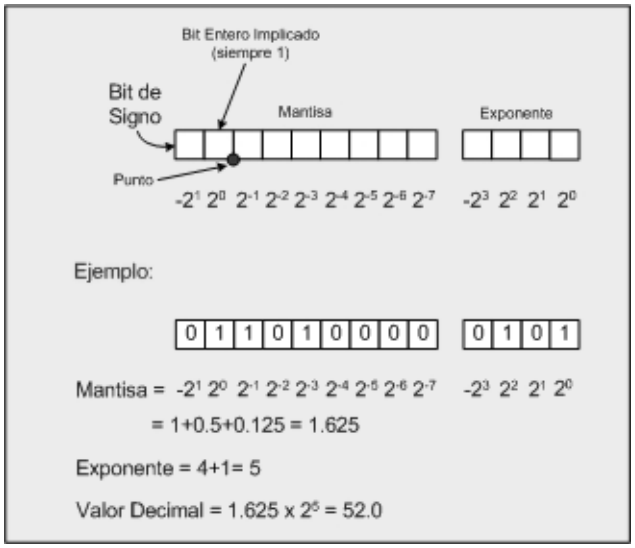


Figura 3.5 Representación de números en punto flotante.

3.1.1.5 ANCHO DE PALABRA DE DATOS

Como se dijo anteriormente, la clasificación de los DSP's se realiza con base en el tipo de aritmética que utilizan para realizar los cálculos matemáticos dividiéndose en DSP's de punto fijo y DSP's de punto flotante. Dentro de cada grupo se clasifican, además, según el ancho de su palabra de datos.

El CPU de los procesadores de punto fijo requiere un hardware más simple que el de punto flotante. Esto se traduce en una reducción del costo unitario del DSP, haciéndolos idóneos para aplicaciones de gran consumo que no requieran alta resolución. Esta simplicidad del CPU también reduce el consumo del dispositivo y su tamaño, un aspecto sumamente interesante para aplicaciones portátiles como los teléfonos móviles. La utilización de un CPU poco sofisticado permite liberar área del circuito integrado para incluir bancos internos de memoria RAM de mayor tamaño o incluso bancos EPROM o FLASH, donde se graba el código de la aplicación. Además, suelen disponer de un conjunto de periféricos más variados. De hecho, los procesadores destinados a aplicaciones específicas (control de motores, sistemas de tratamiento de voz, etc.) son procesadores de punto fijo.

La anchura de la palabra de datos puede ser de 16, 24, 32 o 64 bits. Esto tiene una importante repercusión en el costo, ya que influye poderosamente en el tamaño y número de terminales del dispositivo, y en los bancos de memoria externa conectados

al mismo procesador. Por lo tanto, generalmente los diseñadores intentan utilizar el dispositivo con el menor ancho de palabra que su aplicación pueda tolerar.

Los DSP's de punto flotante son dispositivos de alta escala, cuyo CPU dispone de hardware específico para operar con datos de punto flotante. La anchura de la palabra con frecuencia es de 32 bits; si bien el uso de 32 bits les permite operar en el interior del CPU con datos de hasta 40 bits, esto no es obstáculo para que también puedan operar con datos en punto fijo. El área ocupada por el CPU en este tipo de DSP's es mayor que en el caso de los de los de punto fijo. Por este motivo, la variedad de los periféricos que integran es menor, tratándose en la mayoría de los casos de periféricos (puertos serie y paralelo, DMA) utilizados en la comunicación con elementos externos (convertidores A/D y D/A). Los DSP's más sofisticados disponen de puertos de comunicación que permiten el montaje de redes para un procesamiento en paralelo.

Cada tipo de procesador es ideal para un ámbito de aplicaciones. Los procesadores de 16 bits de punto fijo son adecuados para sistemas de voz, como teléfonos, ya que trabajan con el intervalo relativamente estrecho de las frecuencias de sonido. Las aplicaciones estero de alta fidelidad tienen un intervalo de frecuencias más amplio, de forma general, los requerimientos mínimos para este tipo de tareas serían un ADC de 16 bits y un procesador de 24 bits de punto fijo, de esta forma se proporciona un intervalo suficientemente amplio para obtener la señal de alta fidelidad y para poder manipular los valores que se obtienen al procesar la señal. El procesamiento de imágenes, gráficos en 3D y simulaciones científicas tienen un intervalo dinámico mucho más amplio, por lo que se podría necesitar DSP's de 32 o 64 bits con aritmética de punto flotante.

Para nuestro caso seleccionamos un procesador de 32 bits de punto fijo, con una frecuencia de reloj de 200 MHz y de bajo consumo de energía.

3.1.1.6 PARALELISMO

Otra clasificación de los DSP's es atendiendo al paralelismo de éstos, entendiendo como tal la posibilidad de ejecutar múltiples instrucciones de forma concurrente. Este paralelismo puede ser explícito o implícito. En la figura 3.6-a se observa un DSP con paralelismo explícito que integra varios CPU's en un mismo encapsulado, los cuales se

comunican entre sí por medio de una memoria compartida interna. En este tipo de paralelismo es responsabilidad del usuario el reparto del código a ejecutar por parte de cada uno de los CPU's, lo que hace que sea una tarea muy tediosa de programar las rutinas de comunicación en este tipo de dispositivos.

Un DSP con paralelismo implícito como el de la figura 3.6-b dispone de un solo CPU con múltiples unidades funcionales (por ejemplo, varias ALU, multiplicadores y conjuntos de registros) de forma que se puedan ejecutar en paralelo varias instrucciones, cada una de ellas sobre una unidad funcional. Para ello disponen de la palabra de instrucción de gran tamaño (Very Long Instruction Word de 256 bits), en la que se empaquetan varias instrucciones individuales. La planificación de qué instrucciones se van a ejecutar en paralelo en cada momento la realiza el propio compilador, en un proceso totalmente transparente para el programador. Este tipo de procesadores es sin duda la vanguardia de la tecnología de los DSP.

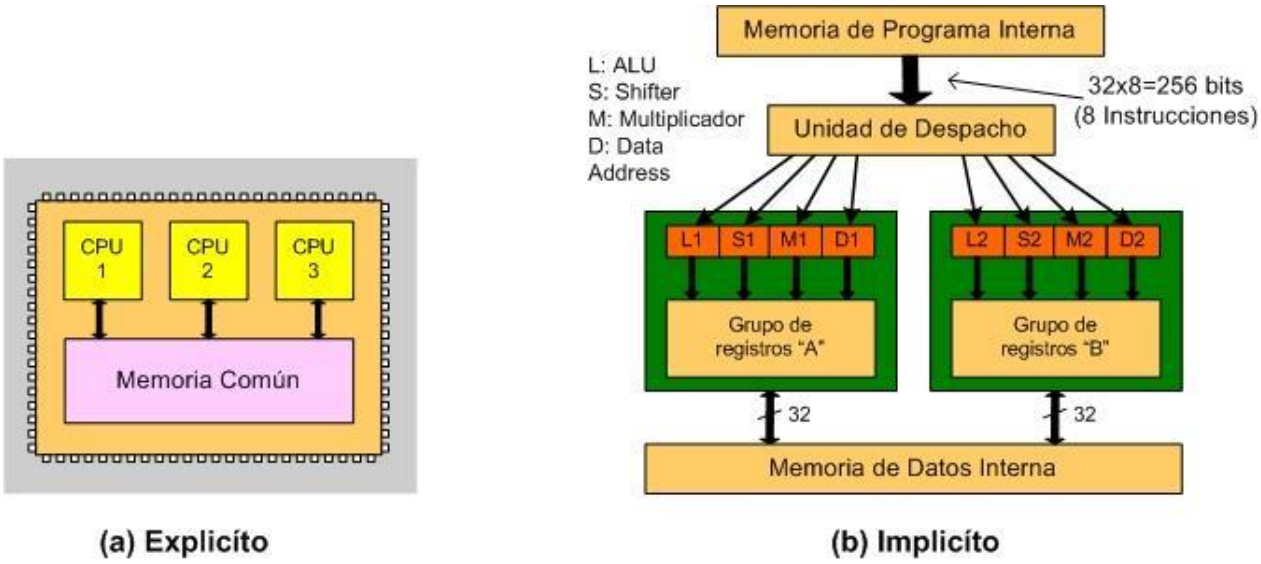


Figura 3.6 Tipos de DSP según su paralelismo.

3.1.2 Arquitectura

Para incrementar la velocidad de trabajo y optimizar la implementación de los algoritmos de procesamiento digital de señales, el CPU de los DSP's dispone de unidades computacionales específicas que pueden trabajar en paralelo, logrando la ejecución repetitiva de este tipo de operaciones en un solo ciclo de instrucción.

La figura 3.7 muestra la ruta de datos típica, en este caso se trata del procesador de punto fijo DSP5600x de Motorola, de 24 bits. Muchos de los DSP's del mercado tienen una arquitectura interna similar a la del DSP5600x, donde las interconexiones y operaciones pueden variar.

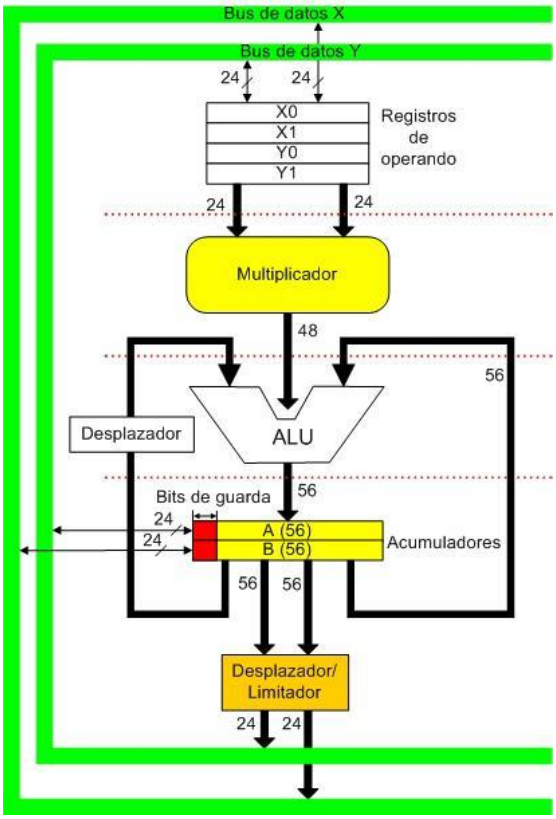


Figura 3.7 Ruta de datos representativo de un DSP de punto fijo (DSP5600x).

Los DSP's incluyen los siguientes componentes:

- *Multiplicador.*

  Se encarga de realizar una operación esencial en los DSP's, la multiplicación, la cual es realizada en un solo ciclo máquina. Mientras el multiplicador produce un nuevo resultado por ciclo de instrucción, el pipelining interno del multiplicador puede tener un retardo de más de un ciclo, este retardo se denomina latencia.

- *Unidad Aritmética Lógica (ALU)*

  La ALU del DSP es la encargada de ejecutar las operaciones aritméticas básicas como suma, resta, incremento, etc. Además implementan operaciones lógicas tal

como: *and, or, y not*. La estructura básica de una unidad aritmética lógica consiste en utilizar multiplexores con tantas entradas como operaciones realice dicha ALU. Generalmente se sobredimensiona el número de bits de la ALU, añadiéndose un cierto número de bits extra, denominados bits de guarda, para realizar la suma de los términos del producto, esto debido a la posibilidad de desborde de datos al realizar las operaciones de acumulación.

- *Desplazador*

  Como resultado de la multiplicación y acumulación hay un crecimiento en la anchura de bits del resultado aritmético. Un desplazador en la ruta de datos facilita la selección de bits de resultado para pasarlos a la siguiente etapa de procesamiento, escalando (multiplicando) su entrada por una potencia de dos (2^n). El desplazador se usa para pre-escalar un operando en la memoria de datos o en el acumulador antes de una operación en la ALU, o también para post-escalar el valor del acumulador antes de almacenarlo en la memoria de datos. Este componente es implementado mediante lógica combinacional (denominada barrel shifter). Esto permite realizar desplazamientos en un solo ciclo de instrucción, independientemente del número de bits a desplazar. Algunos procesadores tienen múltiples desplazadores con diversas capacidades en diferentes lugares de la ruta de datos.

- *Desborde y saturación*

  Los DSP's incorporan técnicas de saturación aritmética agregando circuitos que al detectar una situación de desborde durante la suma de dos números positivos, sustituyen la salida errónea por el máximo valor positivo a representar. De igual manera, si la situación de desborde se da durante la suma de dos valores negativos proporciona como salida el máximo valor negativo a representar.

  De esta forma se consigue un comportamiento similar al de un circuito analógico en saturación. La saturación aritmética se puede realizar mediante una instrucción especial o automáticamente. Algunos fabricantes denominan limitador al módulo que implementa la saturación aritmética.

- *Generador de direcciones de dato (DAG)*

  Otra característica que permite aumentar la velocidad de procesamiento aritmético en los DSP es la disponibilidad de una o más unidades generadoras

de direcciones de datos (DAG, Data Address Generator), las cuales calculan la nueva dirección necesaria para el acceso a los operandos. Estas unidades constan de una unidad aritmética específica que opera en paralelo al resto de las unidades funcionales, y de un conjunto de registros que proporcionarán la dirección base y el desplazamiento necesario para el cálculo de la nueva dirección. Las DAG de varios DSP permiten implementar modos de direccionamiento pensados especialmente para la realización rápida de la transformada discreta de Fourier.

Generalmente el multiplicador se integra con un sumador, formando una unidad multiplicador-acumulador o MAC. Ambos elementos operan en paralelo para efectuar la operación MAC, siguiendo un proceso segmentado en el que, mientras el multiplicador realiza el producto de dos operandos, la ALU acumula el resultado del producto anterior. Esta unidad MAC puede operar con números enteros o en punto flotante.

Muchas aplicaciones de los DSP's involucran la acumulación en series de valores. Esto sucede, por ejemplo, en algoritmos de filtrado, cuando los elementos de una serie de datos se multiplican por coeficientes y los productos resultantes se suman. Una situación de desborde se manifiesta cuando al sumar dos números positivos se obtiene un resultado negativo y viceversa.

Al repetir las operaciones de acumulación de los productos puede ocurrir el desborde de bits, obteniéndose resultados erróneos. Por lo tanto se sobredimensiona la ALU y también los registros donde se almacena el valor del acumulador.

3.1.3 Juego de instrucciones

El juego de instrucciones es un factor clave a la hora de determinar no sólo qué operaciones son posibles de realizar en un procesador, sino también cuando su uso es natural y eficiente. Las instrucciones controlan y operan los datos del CPU, cómo se leen y almacenan estos en memoria, etc.

Las operaciones a realizar en el procesamiento digital de señales son variadas, y la mayoría de las veces se limitan a multiplicaciones y sumas. Los DSP's disponen de un juego de instrucciones optimizado para este tipo de aplicaciones, con un número de

instrucciones reducido, implementándose frecuentemente como microprocesadores con arquitectura RISC. Los beneficios de estas instrucciones especiales son dobles: por una parte permiten un código más compacto que requiere menos espacio en memoria y por otra parte incrementan la velocidad de ejecución de algoritmos específicos del procesamiento de la señal.

Estas instrucciones están optimizadas para las aplicaciones mencionadas, haciendo uso del paralelismo interno del CPU. Así, una misma instrucción puede realizar el producto de dos números almacenados en las posiciones de memoria especificadas por sus respectivos registros de direcciones, acumular el resultado con el contenido de otro registro de datos e incrementar los registros de direcciones.

Los algoritmos de DSP's más frecuentes, como son convolución, correlación, multiplicación de matrices, etc., se realizan mediante la ejecución repetitiva de una misma instrucción o conjunto de instrucciones denominadas lazos internos o núcleos de algoritmo.

La penalización en la ejecución que introducen las instrucciones de salto es especialmente importante en núcleos pequeños. Puesto que la mayoría de estos bucles se ejecutan en un número fijo de veces, el procesador debe utilizar un registro para almacenar el índice del lazo. El CPU debe utilizarse para incrementar el índice y comprobar si se verifica la condición de repetición del bucle. Si es así, se vuelve al comienzo del lazo mediante una instrucción de salto condicional. Todos estos pasos penalizan la ejecución del lazo y utilizan registros innecesariamente.

Para solventar estos problemas, los DSP utilizan lazos de hardware (zero overhead looping). Éstos son estructuras de control especiales que repiten una sola instrucción, o conjunto de instrucciones, un determinado número de veces. Son especialmente efectivos para el caso de lazos de instrucción única, ya que se precisa traer la instrucción de memoria una única vez, liberando los buses para realizar otras operaciones como el acceso a datos o coeficientes. La diferencia fundamental con los lazos software es que un lazo hardware no pierde tiempo en incrementar o disminuir un registro, para comprobar si se ha llegado al final del lazo o saltar al origen de éste.

3.1.4 Arquitectura de memoria

En las aplicaciones de procesamiento digital de señales se procesa un gran volumen de datos. Debido a la limitación del número de registros disponibles en el CPU, normalmente estos datos residen en memoria. Para poder realizar las operaciones MAC en un solo ciclo de instrucción es preciso que los operandos utilizados estén disponibles en el momento de ejecutar dicha instrucción. Normalmente la acumulación de productos se realiza sobre un registro del CPU, con lo que el almacenamiento del resultado final en memoria no se considera parte del núcleo del algoritmo. Es decir, la ejecución de una instrucción MAC implica la realización de tres accesos a memoria:

- Un primer acceso de búsqueda de código de la instrucción a ejecutar.
- Dos accesos para la lectura de los operandos del producto.

Teniendo en cuenta estas condiciones, los DSP's estructuran su memoria de forma que se pueda obtener la instrucción a ejecutar y sus operandos desde memoria al ritmo que los demanda el CPU. De hecho, una de las características distintivas de los DSP's es la forma en que organizan la memoria, siendo ésta un factor crítico en las prestaciones del procesador. Dicha arquitectura está orientada a posibilitar la realización simultánea, en un solo ciclo de instrucción, de los tres accesos anteriormente mencionados.

La arquitectura de memoria utilizada por los DSP's comúnmente es la Harvard, la cual se muestra en la figura 3.8, esta arquitectura dispone de dos espacios de memoria independientes, uno para almacenar el código a ejecutar y el otro para los datos. Cada uno de estos espacios dispone de su propio grupo de buses, con lo cual es posible acceder simultáneamente a ambos espacios. La arquitectura Harvard original restringe el uso a lo que se destina cada uno de los espacios de memoria (almacenamiento de código o de datos). Esta solución no es del todo adecuada, ya que en instrucciones de tipo MAC es preciso acceder a dos operandos en memoria, manteniéndose por tanto el cuello de botella en los accesos a memoria de datos.

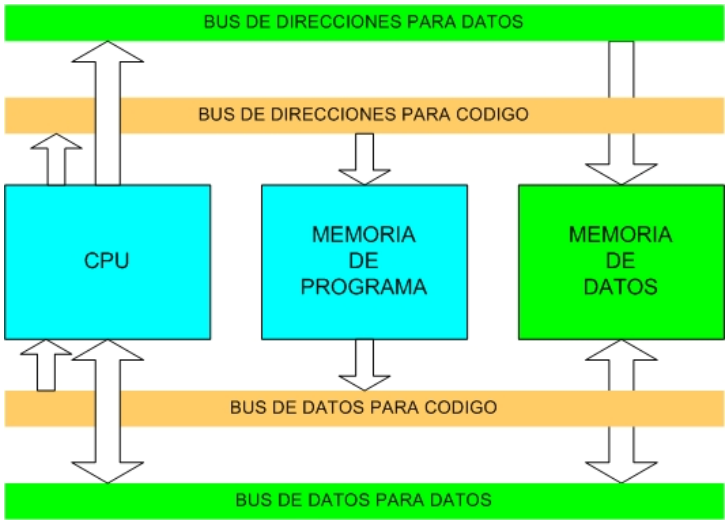


Figura 3.8 Arquitectura Harvard.

En la figura 3.9, esta restricción inicial se ha flexibilizado dando lugar a la arquitectura Harvard modificada, en la cual se permite el almacenamiento de código y datos en el espacio de memoria de programa. Puesto que los DSP suelen disponer de módulos de repetición de instrucciones, una vez obtenido el código de la instrucción MAC, el espacio de memoria de programa queda disponible para obtener uno de los operandos. Por tanto, a excepción de la primera vez que se ejecuta dicha instrucción, las restantes completan su ejecución en un solo ciclo de instrucción.

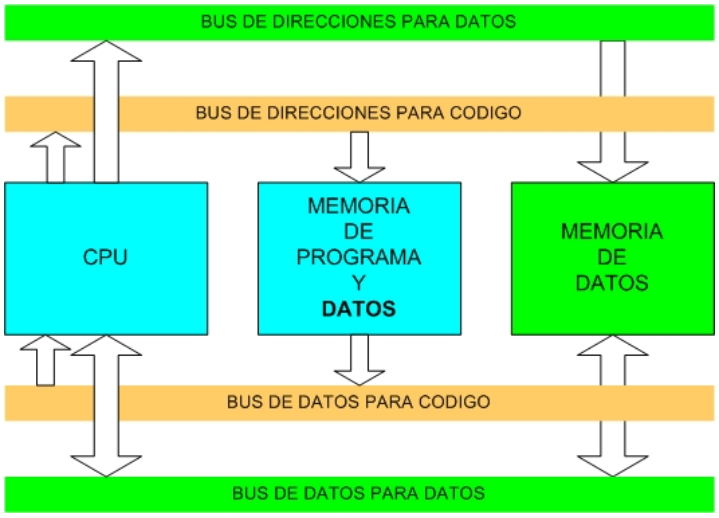


Figura 3.9 Arquitectura Harvard modificada.

Otros enfoques eliminan la latencia introducida durante la ejecución de la primera instrucción MAC desarrollando el concepto de la arquitectura Harvard. Para ello dividen

a su vez la memoria de datos en dos bancos de memoria, cada uno de los cuales dispone de su propio juego de buses de datos y direcciones. Así se dispone de un banco de memoria de programa y dos bancos de memoria para datos, denominados X e Y, respectivamente.

Estas tres memorias permiten al procesador realizar tres accesos independientes por cada ciclo de instrucción (Figura 3.10):

- Una búsqueda de instrucción.
- Un acceso de datos al banco X
- Un acceso de datos al banco Y

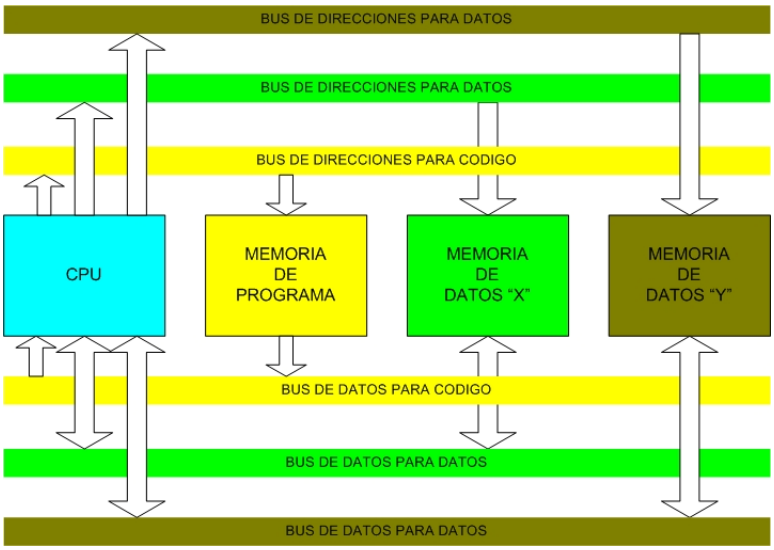


Figura 3.10 Arquitectura Harvard mejorada.

En el caso de los DSP más recientes, desaparecen las restricciones referentes al uso al que se destina cada uno de los espacios de memoria. En su lugar se dispone de un único espacio compuesto por varios bancos de memoria independientes, cada uno con su propio grupo de buses. Es responsabilidad del programador distribuir cada una de las secciones del programa sobre distintos bancos para aprovechar al máximo el paralelismo del procesador.

Puesto que el aumento de múltiples buses de memoria fuera del circuito integrado es costoso, los DSP generalmente proporcionan un único conjunto de buses externo. Los procesadores con múltiples bancos de memoria normalmente

proporcionan una pequeña cantidad de memoria interna para cada uno de los bancos. Aunque los bancos de memoria pueden ampliarse externamente, no se pueden realizar accesos externos múltiples en paralelo, debido a la ausencia de un segundo grupo de buses para la memoria externa. Por tanto, si se precisan múltiples accesos a memoria externa durante la ejecución de una instrucción, la ejecución de ésta se prolongaría a lo largo de varios ciclos de instrucción ya que los accesos a memoria se realizarían de forma secuencial.

Existen otras alternativas no excluyentes al empleo de la arquitectura Harvard, que permiten aumentar el ancho de banda de la memoria. Una de ellas se basa en el uso de memorias de acceso múltiple. Se trata de memorias lo suficientemente rápidas para permitir varios accesos secuenciales por cada ciclo de instrucción a través de un único grupo de buses, o bien utilizar memorias multipuerto que permitan varios accesos concurrentes a memoria sobre dos o más grupos de buses independientes, como se muestra en la figura 3.11.

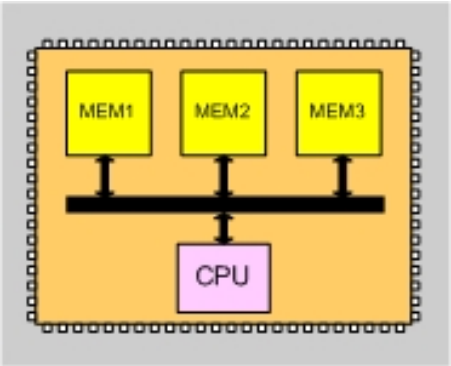


Figura 3.11 DSP con varios bancos de memoria internos.

Tanto las memorias rápidas como las multipuerto se integran en el propio encapsulado del DSP. La capacidad de los bancos de memoria de acceso múltiple suele ser reducida debido a la limitación del área disponible del circuito integrado.

Para completar en un solo ciclo la ejecución de la instrucción MAC (incluso el almacenamiento en memoria del resultado de la acumulación) es preciso distribuir adecuadamente las distintas partes del programa (código, operandos y resultados) de forma que no se sobrepase el ancho de banda de cada banco. En caso contrario, la ejecución del programa se volvería más lenta. Finalmente, otra de las alternativas para

aumentar el ancho de banda de la memoria consiste en el uso de memorias caché de programa, las cuales son pequeñas memorias que se agregan dentro del núcleo del procesador, y que reducen la necesidad de acceder a memoria en la fase de búsqueda de instrucciones. Eliminar este acceso permite un posible acceso que se empleará para leer o escribir un dato, o bien puede eliminar los retardos asociados con una memoria de programa externa más lenta.

Respecto a la interface externa con memoria de un DSP, ésta puede caracterizarse básicamente mediante tres propiedades; el número de puertos disponibles, la sofisticación y flexibilidad de los mismos, y los requerimientos temporales.

Aunque cuenten con varios bancos independientes de memoria internos, la mayoría de los DSP's disponen de un único conjunto de buses externo. Es así porque extender los buses al exterior implica encapsulados con un número de terminales muy elevado, lo que incrementaría notablemente el precio final. Por ello, es imposible realizar varios accesos a posiciones externas en un mismo ciclo de instrucción. La duplicidad de buses externos solo aparece en dispositivos de alta escala.

3.1.5 Periféricos integrados e interfaces I/O

La mayor parte de los DSP integran en el propio encapsulado periféricos de gran versatilidad que le permiten comunicarse con dispositivos externos, tales como un ADC, un DAC, otros procesadores DSP o microprocesadores. Este es un aspecto de gran importancia y que los distingue de los procesadores de propósito general. De entre los más frecuentes se pueden encontrar:

- **Temporizadores**. Normalmente todos los procesadores incluyen estos dispositivos, que suelen admitir como entrada un reloj interno o externo. Se utilizan para la generación de interrupciones periódicas que sincronizan el proceso de muestreo.
- **Puertos Serie**. Las características y complejidad de estas interfases varían de un procesador a otro, pero es habitual encontrarlos en la mayoría de ellos. Suelen utilizarse para comunicarse con circuitos que incluyen conversores A/D y D/A denominados "codecs". Esto permite alejar la parte analógica del procesador, eliminando problemas de ruido. De hecho, la mayoría de los

fabricantes de los DSP diseñan dispositivos de este tipo con una interfase que permite la conexión directa con el puerto serie. No es raro encontrar también DSP que permiten la transmisión del código de arranque en sistemas sin memoria ROM.

- **Puerto Paralelo**. Un puerto paralelo recibe y envía múltiples datos de entre 8 y 16 bits al mismo tiempo. Estos trasmiten la información mucho más rápido que un puerto serie; el inconveniente es que se requieren más pines para esta acción. Para ahorrar pines en la configuración de un puerto paralelo de comunicación con dispositivos externos, el DSP hace uso del bus de datos principal como puerto paralelo.
- **Controlador DMA**. Este tipo de periférico permite efectuar transferencias de forma rápida, sin intervención del procesador. Esto es especialmente útil en aplicaciones en las que el volumen de datos a manejar es considerable; por ejemplo, procesamiento digital de imagen. En tal caso el DMA se utiliza para llevar bloques de datos desde una memoria externa de gran capacidad a la memoria interna, con objeto de que se puedan procesar a mayor velocidad.
- **Interface con un Host (HPI).** En algunos casos los DSP funcionan como coprocesadores matemáticos, formando parte de un sistema global controlado por un procesador de propósito general. Para comunicarse con este procesador host, algunos DSP incorporan un puerto paralelo bidireccional de 8 o 16 bits, que podría incluso estar diseñado específicamente para comunicarse con un bus estándar, como el ISA o el PCI. Las comunicaciones a través de este puerto suelen realizarse mediante DMA de manera que sea posible transferir datos entre memoria y el puerto, sin intervención del procesador. El control del puerto se realiza mediante instrucciones específicas. En ocasiones esta interface permite incluso el control del funcionamiento del DSP en tareas tales como; forzar la ejecución de rutinas de tratamiento de interrupción, acceder a los registros internos del DSP o a su memoria, e incluso realizar la carga del código a ejecutar (bootstraping).
- **Puertos de Comunicación.** Se trata de puertos paralelo diseñados para comunicación entre los DSP del mismo tipo, y que pueden conectarse en red para implementar un sistema multiprocesador. Puesto que la anchura de estos

puertos (8 bits) es menor que el tamaño de la palabra de datos de los DSP que los incluyen (32 bits), los puertos disponen de FIFOs para la fragmentación y reensamblado de los datos que se trasmiten a través de ellos. La comunicación a través de estos puertos suele estar asistida por DMA.

- **Puerto de I/O (GPIO).** Estos grupos de pines en un DSP pueden habilitarse como entradas o salidas y se usan para propósitos de control, aunque también se pueden usar para transferencia de datos.

Los DSP diseñados para aplicaciones muy específicas, como el control de motores o sistemas de potencia, disponen de periféricos concretos para dichas aplicaciones, como pueden ser: generadores de señal PWM, temporizadores especiales para la implementación de tacómetros digitales, convertidores A/D, D/A, etc. En general se trata de DSP de baja escala, ya que el área utilizada del circuito integrado para estos periféricos limita el tamaño de los bancos de memoria interna y la sofisticación del CPU.

En la figura 3.12 se describe brevemente como se configura internamente un DSP, el cual contiene una gran variedad de periféricos: controlador de DMA, puertos serie y timers. En el diagrama se incluye la existencia de una arquitectura Harvard. Así aparecen múltiples bancos de memoria interna. Además, esta arquitectura se manifiesta también en el exterior del dispositivo con dos buses externos: bus principal y bus de expansión.

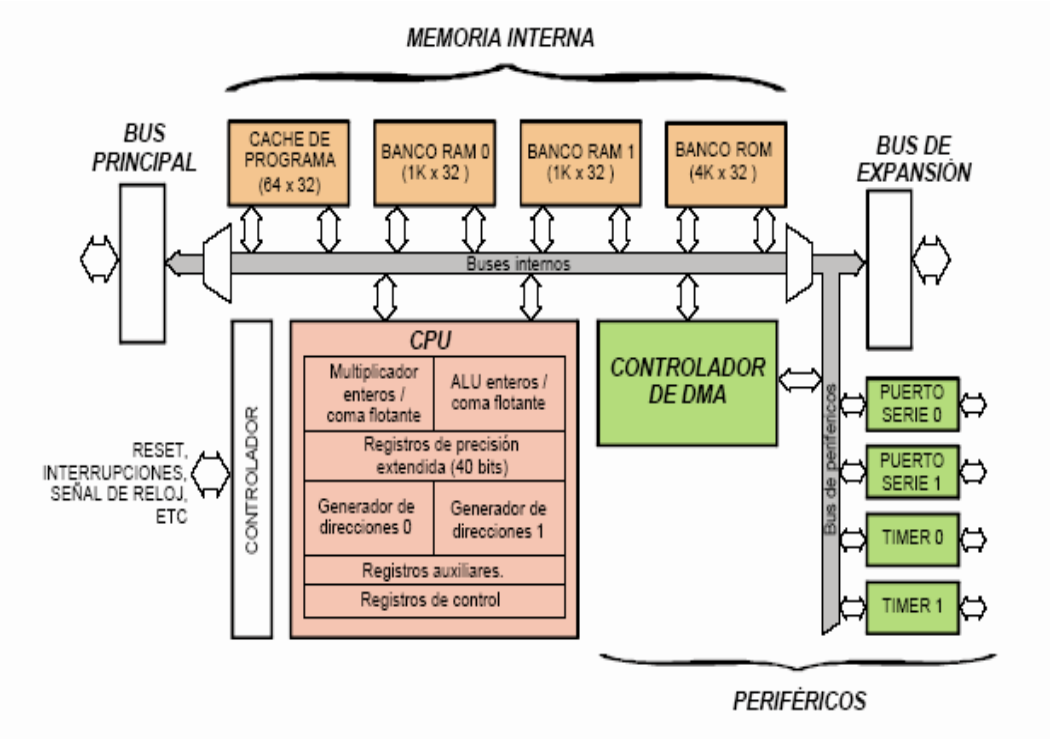


Figura 3.12 Arquitectura del TMS320C5x de Texas Instruments.

3.1.6 Criterios de selección

Para un observador casual, todas las arquitecturas de un DSP parecerían muy similares. No así, para un usuario que tiene bien definidas las necesidades y restricciones de una aplicación. Es decir, deben de valorarse las premisas que en algunos casos son muy particulares para determinar las características que debe tener el DSP. Por ejemplo, sí el objetivo es desarrollar un producto y se está limitado en el *tiempo* de lanzamiento al mercado, un DSP de punto flotante podría ser una buena elección, ya que su desarrollo de software es más simple que los dispositivos de punto fijo. Sí el *costo* es la consideración primaria, hay un sinnúmero de DSP's de bajo costo, el problema es que vienen comercializados con límite de rendimiento y el software de desarrollo cambia continuamente. Sí el *consumo* de energía es una restricción clave, entonces la elección de un DSP de bajo consumo de energía, con una arquitectura basada en un núcleo ASIC puede ser la solución para aplicaciones especiales.

En muchas aplicaciones, algunos de los factores mencionados son indispensables, sin embargo es importante identificar la arquitectura que mejor se adapte a las

características de una determinada situación. A continuación se enlistan las características básicas que debe reunir el DSP.

- **El tipo de aritmética utilizada y el ancho de palabra de datos**. La principal diferencia en el uso de DSP de punto fijo o punto flotante radica en cómo son los resultados de las operaciones de multiplicación manipuladas. Los procesadores de punto fijo tienen un hardware más simple que los de punto flotante, tomando en cuenta que éstos últimos son dispositivos más caros y de mayor consumo. Una recomendación para los diseñadores es que deben intentar el uso de un DSP con el menor ancho de palabra que su aplicación puede tolerar. En la figura 3.13 se observa la clasificación de los DSP's por el tipo de aritmética que usan y la anchura de datos.

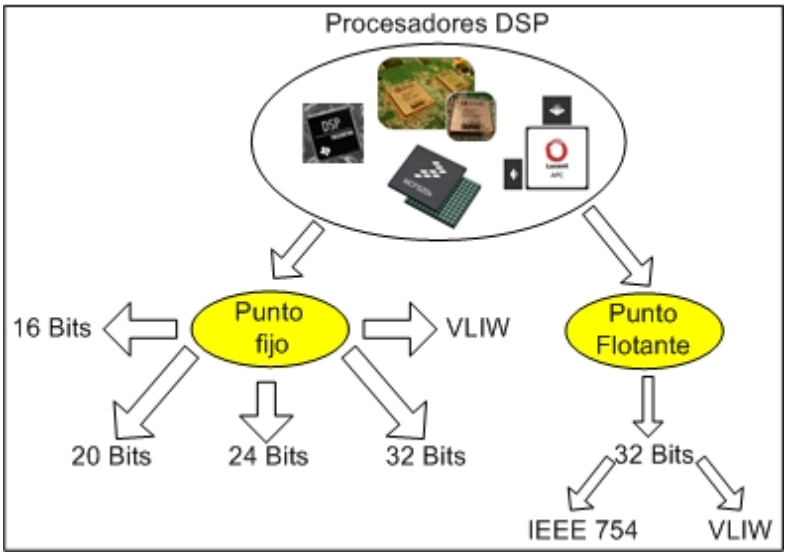


Figura 3.13 Clasificación de los DSP según su representación aritmética.

- **Velocidad.** Algunas arquitecturas claramente tienen ventaja en ciertas áreas. Una de esas áreas muy importante es la velocidad de ejecución ya que los ciclos de tiempo de instrucción pueden variar significativamente en la ejecución de la aplicación. En general, las mediciones del rendimiento de un procesador deben tomarse con cierto escepticismo. En particular, las comparaciones basadas en MIPS y MFLOPS son muy inciertas, por la gran diferencia en el monto de procesamiento que los procesadores pueden lograr con una sola instrucción u operación de punto flotante.

- **Tamaño de Memoria.** Una arquitectura de memoria eficiente es aquella que permite procesar las instrucciones y datos al ritmo que los demanda el CPU. Es deseable que los DSP dispongan de la mayor cantidad de memoria interna, ya que los accesos sobre ésta se realizan a mayor velocidad. La disponibilidad de memoria Flash interna permite reducir la complejidad del sistema.
- **Periféricos integrados.** Esta consideración es una de las más importantes cuando se selecciona un DSP, por lo que se deben definir claramente las aplicaciones. La inclusión de periféricos o interfaces que le permitan comunicarse con el exterior en un mismo dispositivo puede tener un impacto significativo en el costo del procesador. Los DSP de punto fijo tienen a su disposición un conjunto de periféricos más variado que los de punto flotante, lo que los hace más eficientes en aplicaciones específicas. Por ejemplo la familia C2000 de Texas Instruments está enfocada a aplicaciones de control de motores, e incorporan a sus DSP periféricos tales como: Timers, PCI, DAC, ADC, GPIO, PWM, etc.
- **Consumo.** Este es un factor determinante en algunas aplicaciones, tales como las móviles, caso específico el de los teléfonos celulares. Actualmente se encuentran DSP con un voltaje nominal entre 3.0 y 3.3 V. Además, los fabricantes están agregando características de ahorro de energía controlado por medio de software o hardware. Estos modos de ahorro consisten en apagar ciertas secciones del procesador, así que el diseñador debe considerarlos si su aplicación así lo requiere.
- **Costo.** En algunas aplicaciones el DSP podría representar una pequeña fracción del costo total del sistema. En otras, el DSP podría comprometer un gran porcentaje del costo de inversión. En aplicaciones de gran consumo este aspecto puede prevalecer sobre otros que inciden más directamente sobre las prestaciones del DSP.
- **Rango Dinámico.** Es un parámetro que relaciona el tipo de aritmética utilizada y el ancho de la palabra de datos. Se define como la relación que existe entre el máximo y mínimo número que se pueden representar y por supuesto diferente de cero. A este respecto, los procesadores de punto flotante tienen un rango dinámico más amplio lo cual se traduce en una mejor precisión.

- **Soporte y software de desarrollo.** Cuando se selecciona un procesador, es importante considerar que tipo de soporte proporciona el fabricante que ayude al proceso de diseño y problemas que puedan surgir. Tradicionalmente, el software para la programación de los DSP se escribe en lenguaje ensamblador, lenguaje “C”, y debe ser extremadamente eficiente.

3.2 BUS SERIAL UNIVERSAL (USB)

A continuación se abord otra parte importante en el desarrollo presentado en este trabajo, el puerto universal serial, mejor conocido como USB, se dará una introducción a este puerto y se mencionarán los diferentes conceptos importantes para un desarrollo utilizando este protocolo de comunicación.

Como ya se ha mencionado anteriormente, en este trabajo se realiza una implementación de comunicación entre un dispositivo médico y una PC mediante este protocolo.

El poder de las computadoras ha crecido, el número de tareas que se requiere que una computadora realice también ha crecido y el número de dispositivos que requerimos conectar a una PC igualmente ha crecido, por lo anterior se ha buscado formas más eficientes de realizar comunicación desde una PC con dispositivos externos.

A continuación se muestra una tabla, en la cual se muestran diferentes interfaces de comunicación, las cuales se han utilizado para conectar diferentes dispositivos hacia una PC, se puede apreciar que el USB es más flexible que otras interfaces, las cuales tienen un uso específico.

Interfaz	Tipo	Número de dispositivos (incluyendo PC) max	Distancia máxima (ft)	Velocidad Máxima (b/s)	Uso típico
USB 3.0	Serial dual simplex	127 por bus	9 (hasta 49 con 5 hubs)	5G	Almacenamiento, adquisición de datos y transmisión de video.

USB 2.0	Serial half duplex	127 por bus	16(hasta 98 con 5 hubs	1.5M, 12M, 480 M	Teclado, unidades de disco, bocinas, impresoras, cámaras, adquisición de datos.
USB 1.1	Serial half duplex	127 por bus	16	1.5M, 12M	Teclado, unidades de disco, bocinas, impresoras, cámaras, adquisición de datos.
eSATA	Serial	2	6	3G	Disco duros
Eterneth	Serial	1024	1600	10G	Comunicaciones generales de redes
IEEE-1394b (FireWire)	Serial	64	300	3.2G	Video y almacenamiento
IEEE-488 (GPIB)	Paralelo	15	60	8M	Instrumentación
MIDI	Lazo de corriente serial	2	50	31.5k	Música, control de luces
Paralelo	Paralelo	2	10-30	8M	Impresoras, scaners, discos duros
RS-232	Serial asíncrono	2	50-100	20k (115k con cierto hardware)	Modem, ratón, instrumentación
RS-485	Serial asíncrono	32 (algunos chips soportan 256)	4000	10M	Adquisición de datos y sistemas de control

Tabla 3.1 Comparación entre diferentes formas de comunicación e interfaces de PC's.

A continuación se enlistan algunas de las características del USB:

- Diferentes dispositivos de Entrada/Salida pueden ser conectados a la PC mientras ésta se encuentra funcionando.
- Los dispositivos de Entrada/Salida conectados son reconocidos por la PC y por su manejador, y la configuración es realizada automáticamente.
- Todos los dispositivos utilizan el mismo tipo de conector.
- Alta transferencia de datos, hasta 5Gbps para la versión 3.0.
- Hasta 126 dispositivos se pueden conectar por bus presente.
- La alimentación puede ser suministrada por el cable de comunicación, por lo que la mayoría de dispositivos no requieren de fuente de alimentación adicional.
- Eficiente manejo de la potencia, debido a que los dispositivos se colocan en modo de bajo consumo cuando no se están utilizando.

- Existe detección de errores automático y las transacciones son reintentadas para asegurar que los datos sean entregados confiablemente.
- No hay necesidad de abrir la PC o diseñar tarjetas que deben ser instaladas en la PC.

Todo lo anterior es parte de la especificación del bus que empresas han desarrollado para volver este protocolo un estándar [38]. USB, es ahora considerado como la interfaz de comunicación más exitosa de una computadora personal, cada PC actual cuenta con puertos USB que pueden conectar una amplia variedad de dispositivos, como teclados, ratones, controladores de juego, escáneres, impresoras, cámaras, unidades de disco, dispositivos de adquisición de datos, entre otros. USB a su vez es confiable, rápido, versátil, ahorrador de energía, barato y soportado por la mayoría de los sistemas operativos. Pero también tiene algunas cuestiones no tan favorables las cuales también se mencionarán a continuación.

3.2.1 Bases.

El puerto USB (Universal Serial Bus) fue introducido al mundo de la computación para resolver el problema creciente para interconectar dispositivos externos o periféricos a la computadora (PC) [39]. Se requería de alguna manera, no sólo para conectar dispositivos físicamente, sino también permitirles ser identificados y categorizados.

Esta interfaz se ha convertido en la más exitosa, en los últimos años, para una computadora, cada PC reciente cuenta con puertos USB, en los cuáles puedes conectar desde periféricos estándar como dispositivos de interfaz humana (HID) dispositivos de almacenamiento masivo hasta equipos más especializados de adquisición de datos [40], [41] e inclusive diseños propios; siempre y cuando se ajusten a las especificaciones necesarias.

Desde el punto de vista del usuario final, el USB es de fácil uso, tiene transferencias de datos rápidas y confiables, es de bajo costo y bajo consumo de potencia.

Se considera de fácil uso debido a que es lo suficientemente versátil para soportar muchos periféricos; en vez de tener un diferente conector y tipo de cable para cada periférico, una misma interfaz se ocupa para muchos. Además de tener una

configuración automática, fácil conexión, conectores compactos, y actualmente existe una opción para conexión inalámbrica, los dispositivos USB se pueden desconectar y conectar en cualquier momento sin importar si están energizados y sin dañar ni la PC ni el dispositivo; no se requiere de configuración especial por el usuario tal como dirección de puerto, líneas IRQ y en algunas ocasiones no se requiere suministrar alimentación externa al dispositivo, ya que puede ser alimentado por el propio bus.

El bus USB soporta cuatro velocidades: SuperSpeed a 5Gbps, high speed a 480Mbps, full speed a 12Mbps y low speed a 1.5Mbps. La velocidad del bus describe la taza a la que viaja la información sobre éste. Es importante considerar que además de los datos de la aplicación, el bus debe llevar información propia del protocolo como son: estatus, control y revisión de errores.

Otra característica es que múltiples dispositivos pueden compartir un mismo bus. Aunque esta pudiera reflejar una debilidad ya que el rendimiento en la transferencia de los datos para un dispositivo en particular es menor que la velocidad del bus. Por ejemplo para un dispositivo full speed, que es el protocolo utilizado en este trabajo, la velocidad de transferencia de datos para una aplicación es únicamente hasta 1.2MB/s.

Existen hasta el momento 4 especificaciones definidas para este bus. USB 1.0 que soporta low speed, 1.1 que soporta hasta full speed, USB 2.0 soporta hasta high speed y la más reciente USB 3.0 soporta hasta SuperSpeed.

La confiabilidad del USB es aplicable tanto para hardware como para los protocolos de comunicación [42]. Las especificaciones de hardware para los drivers, dispositivos y cables USB aseguran una pequeña interfaz eléctrica que elimina la mayoría del ruido que podría causar errores de datos. Y los protocolos de comunicación habilitan la detección de errores en datos recibidos e informan al transmisor para que éste pueda transmitir los datos automáticamente.

Debido a que la idea de esta interfaz es que el usuario final no tenga ninguna dificultad al momento de hacer uso del dispositivo, ya sea tener que configurar el software o el hardware; es por eso que la dificultad se presenta del lado del desarrollador, ya que en este sentido la complejidad es mayor, para ello es necesario conocer primero como

funciona la interfaz USB tanto del lado del dispositivo externo como del de la computadora.

3.2.2 Evolución de la Interfaz

Antes de la llegada del USB, mucha gente sobre cargaban el uso de sus puertos seriales y paralelos para soportar más de un periférico sobre ese puerto. En 1996, la especificación 1.0 del USB fue liberada para resolver este tipo de problema. USB 1.0 ofreció un bus de comunicación con anchos de banda de 12 Mb/s (full speed) o 1.5 Mb/s (low speed) compartido hasta 127 periféricos. La especificación 1.1 fue liberada en Septiembre del 1998, la cual actualiza y arregla problemas encontrados en la especificación anterior.

En Abril del 2000, fue liberada la especificación 2.0 de USB, la cual arregló algunos problemas identificados en las dos versiones anteriores (full y low speed), además incorporó una nueva tasa de transferencia de 480 Mb/s (high speed). La principal motivación para el desarrollo de esta especificación parte del hecho de que las PC's y sus periféricos han incrementado ampliamente su desempeño y funcionalidad y son capaces de procesar una gran cantidad de datos. USB 2.0 es una evolución natural de este bus, dando un incremento del ancho de banda mientras se mantenga la idea original del USB y manteniendo compatibilidad completa con dispositivos existentes.

En el 2005, con el incremento en el uso de tecnologías inalámbricas, Wireless USB fue introducido para proveer una nueva capacidad de conexión del USB libre de cable. Recientemente, en Noviembre del 2008, fue liberada la especificación del USB 3.0, con un ancho de banda de 5 Gb/s (Super Speed).

El presente trabajo fue desarrollado con el DSP5509A, el cual tiene integrado el USB 1.1, y se desarrolló bajo el ancho de banda de 12 Mb/s (full speed). En su momento se consideró realizar el cambio a otro dispositivo que soportara la especificación 2.0 con ancho de banda de 480 Mb/s, pero debido a los resultados que se obtuvieron no fue necesario este cambio en el momento del desarrollo. Ahora, se deja planteado para un trabajo futuro el continuar con este desarrollo para implementarlo bajo la especificación USB 3.0.

3.2.3 Componentes del bus

USB tiene cuatro tipos de transferencias [43], que más adelante serán analizadas cada una de ellas, y ahora ya también cuenta con 4 velocidades, haciendo a esta interfaz viable para una gran cantidad de periféricos. Desde el lanzamiento del USB 1.0, este puerto cuenta con tipos de transferencias las cuales lo hacen apropiado para intercambiar grandes y pequeños bloques de datos, con y sin limitaciones de tiempo.

Para aplicaciones donde no se permita retardos, USB puede garantizar el ancho de banda [44], tal y como en este trabajo se plantea. Estas habilidades son especialmente útiles bajo el ambiente Windows, donde el acceso a periféricos en tiempo real es un reto. Aunque el sistema operativo, controladores de dispositivos y aplicaciones de software pueden introducir retardos inherentes, USB realiza esto lo más fácilmente posible para lograr transferencias que pueden estar cercanas al tiempo real aun en computadoras.

A diferencia de otras interfaces, USB no asigna funciones específicas a líneas o puerto de señales; ni realiza otras suposiciones acerca de cómo el sistema operativo usará la interfaz. Por ejemplo, en el puerto paralelo, existen líneas como la de control y estatus que ya están definidas solamente para la comunicación con impresoras. Debido a que el USB no realiza tal suposición es perfectamente viable para cualquier tipo de periférico.

Todas las ventajas del USB significan solamente que este puerto es un buen candidato para muchos dispositivos, pero no asegura la completa utilidad para todas las tareas.

Algunos límites del USB pueden ser las restricciones de distancia, soporte en hardware y sistemas operativos antiguos y además no soporta comunicaciones punto a punto ni de broadcasting.

USB fue diseñado como un bus de expansión de una PC, donde los dispositivos a conectarse por este puerto se encuentran relativamente cercanos. Otras interfaces, como RS-232, RS-485, IEEE1394b, y Ethernet, permiten conexiones más largas que USB.

Cada comunicación por USB es entre un host y un dispositivo o periférico, ver figura 3.14.

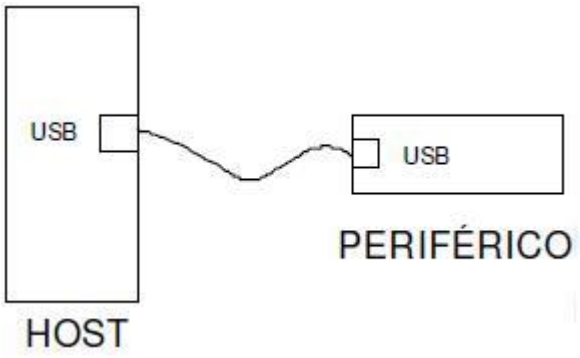


Figura 3.14 Conexión de un Host al periférico (dispositivo) por USB.

Donde el host es una PC o un dispositivo con controlador de host en hardware y el cual representa al mecanismo USB, con un respectivo hardware y firmware de comunicación.

Por otra parte un dispositivo es aquel que se añade a un host como periférico y el cual tiene un controlador de dispositivo por hardware y un firmware para la comunicación.

También podemos tener el siguiente esquema de conexión, figura 3.15.

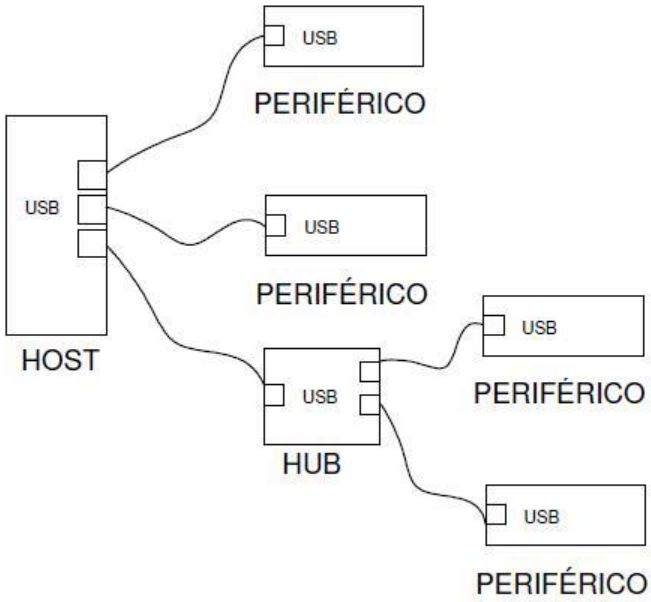


Figura 3.15 Conexión en cascada de varios dispositivos a un host.

Si observamos la figura 3.15, podemos visualizar que en ningún momento se conecta un host con otro directamente ni tampoco un dispositivo con otro dispositivo, esta es una restricción específica de la comunicación por USB, la cual no permite comunicar dos dispositivos con la misma funcionalidad. A diferencia de otras interfaces tal como IEEE-1394, la cual permite comunicación directa entre un dispositivo y otro.

También podemos apreciar, en la figura 3.15, algunas terminales llamadas HUB que básicamente sirven para expandir el número de entradas al Host, los cuales, en gran medida, no afectarán a nuestras aplicaciones. El número máximo de periféricos y HUBs que se pueden conectar a un mismo Host es de 127 (incluyendo el HUB raíz) y en cascada se pueden colocar no más de 5 HUB externos.

Para la restricción de conectividad entre dispositivos de la misma funcionalidad, el protocolo USB provee una solución parcial con la opción llamada USB On-The-Go [45]. Un dispositivo de este tipo puede funcionar tanto como dispositivo y como host con compatibilidad limitada. Dos hosts pueden comunicarse entre ellos solamente vía un cable especial llamado “puente”, el cual contiene dos dispositivos USB con un buffer compartido.

Otra restricción de comunicación se menciona a continuación, donde el USB no soporta envío de datos simultáneamente a múltiple dispositivos desde el host. Por el contrario un host debe enviar los datos a cada dispositivo individualmente conectado, a diferencia de la comunicación por IEEE1394 o Ethernet las cuales si soportan el broadcasting.

Cuando un dispositivo es añadido al host, el dispositivo debe responder a peticiones estándar enviadas por el host durante la enumeración, que más adelante se explicará.

Desde la perspectiva del desarrollador, la complejidad del protocolo es un reto importante. Un periférico USB, es un dispositivo inteligente que sabe cómo responder a peticiones y eventos sobre el bus. Los chips varían en que tanto soporte de firmware requieren para llevar a cambio comunicaciones por USB. En la mayoría de los casos, para programar un dispositivo USB, se requiere conocer lo suficiente sobre protocolos de USB o reglas de intercambio de datos sobre el bus. Del lado de la PC, el manejador (driver) del dispositivo aísla a los programadores de la aplicación tener que conocer

muchos de los detalles del USB, pero los desarrolladores del manejador del dispositivo requieren ser familiares con los protocolos del USB y las responsabilidades del driver.

Las aplicaciones USB no pueden solo leer y escribir a una dirección de puerto, ni los dispositivos pueden solo tener una serie de entradas y salidas para leer y escribir directamente, tal y como sucede con el RS232 y el puerto paralelo. Para accesar a un dispositivo USB, las aplicaciones deben comunicarse con un manejador del dispositivo (driver), el cual inicia las comunicaciones con la capa más baja del USB y que a su vez esta maneja las comunicaciones sobre el bus. El dispositivo debe implementar los protocolos que habilitan a la PC para detectar, identificar y comunicarse con el dispositivo.

3.2.3.1 FUNCIONES DEL HOST

Para comunicarse con dispositivos USB, una computadora requiere de hardware y software que soporten funciones de Host. El hardware consiste de un controlador host de USB y un concentrador (root hub) con uno o más puertos USB. El software es típicamente un sistema operativo que permite a los controladores del dispositivo comunicarse con controladores de bajo nivel los cuales accedan al hardware del USB.

El host está encargado del bus y debe saber que dispositivos están en el bus y las capacidades de cada uno de éstos. El host debe hacer también lo mejor para asegurar que todos los dispositivos sobre el bus puedan enviar y recibir datos como los necesiten. Un bus podría tener muchos dispositivos, cada uno con diferentes requerimientos, todos esperando transferir datos al mismo tiempo.

Afortunadamente, el hardware del Host y los controladores en Windows y otros sistemas operativos realizan mucho del trabajo sobre el manejo del bus. Por lo que cada dispositivo conectado al Host debe tener un controlador de dispositivo asignado, el cual permita que las aplicaciones desarrolladas puedan comunicarse con el dispositivo sin ningún problema.

Por lo tanto, las aplicaciones en el host no tienen que conocer detalles específicos del hardware para comunicarse con los dispositivos. Todas las aplicaciones lo que tienen que hacer es enviar y recibir datos usando funciones estándar del sistema operativo u

otros componentes de software. Frecuentemente la aplicación no tiene que preocuparse si el dispositivo usa USB u otra interfaz, esto es trabajo del controlador del dispositivo.

El host realiza cada una de las siguientes tareas:

- Detectar dispositivos.- Al añadir un dispositivo USB al host, éste debe ejecutar un proceso llamado “enumeración”, en el cual, el host determina que velocidad del bus usar, asigna una dirección y solicita información adicional. Cuando un dispositivo es añadido o removido, el concentrador informa al host de dicho evento, y el host enumera al dispositivo añadido o lo remueve de su lista de dispositivos disponibles para aplicaciones.
- Manejar el flujo de datos.- Muchos dispositivos podrían requerir transferir datos al mismo tiempo. El controlador del host divide el tiempo disponible en intervalos y le otorga a cada transmisión una porción del tiempo disponible. Durante la “enumeración”, un controlador de dispositivo solicita el ancho de banda para las transferencias, si el ancho de banda no está disponible, el controlador puede solicitar una porción más pequeña del ancho de banda o esperar hasta que el ancho de banda solicitado esté disponible.
- Checar errores.- Cuando se transfieren datos, el host añade bits de chequeo de errores. Al recibir los datos, el dispositivo realiza cálculos sobre los datos y compara el resultado con los bits de chequeo de errores recibidos. Si los resultados no empatan, el dispositivo le informará al host de esto y por lo tanto el host debería retransmitir los datos. De la misma manera el host checará errores de los datos recibidos desde los dispositivos. USB también soporta un tipo de transferencia, la cual no checa errores para usarla con datos tales como audio en tiempo real, para garantizar una taza de transferencia constante. En el presente desarrollo se trabajará con este tipo de transferencia y con el tipo bulk, más adelante se explicarán los diferentes tipos de transferencias disponibles por USB.
- Si una transmisión falla después de muchos intentos, el host puede informar al controlador del dispositivo del problema y el controlador puede notificar a la aplicación y así se puede tomar una acción necesaria.

- Manejo de potencia.- Además de los cables de datos, un cable USB tiene cables para una fuente de +5V y su tierra. Algunos dispositivos podrían ser alimentados por el bus. El host provee potencia a todos los dispositivos añadidos y trabaja en conjunto con los dispositivos para conservar la energía cuando sea posible. Los dispositivos sólo podrán demandar hasta 500 mA del bus USB, y aquellos dispositivos alimentados por un controlador de host alimentado por batería sólo pueden demandar 100mA al bus. Para ahorrar energía cuando el bus este en estado “idle”, un host puede solicitarle a los dispositivos entrar a un estado de bajo consumo de energía y reducir el uso de la corriente del bus.

3.2.3.2 FUNCIONES DEL DISPOSITIVO

Cuando el host inicia las comunicaciones, el dispositivo debe responder. Pero los dispositivos también tienen tareas que son únicas. El hardware del dispositivo típicamente maneja muchas responsabilidades de las funciones del dispositivo. La cantidad de firmware a desarrollar varía dependiendo del soporte de cada arquitectura de chip utilizado.

Los dispositivos deben realizar las siguientes tareas especificadas:

- Detectar comunicaciones.- Los dispositivos deben detectar comunicaciones enviadas a la dirección de un dispositivo que se encuentra en el bus. Éste almacena los datos recibidos en un buffer y retorna un código de estatus o envía los datos requeridos desde un buffer o un código de estatus. En casi todos los dispositivos, estas funciones son realizadas dentro del hardware y no requiere más código que preparar los buffers para enviar o recibir los datos. El firmware no tiene que tomar ninguna otra acción o realizar decisiones hasta que el dispositivo haya detectado una petición de comunicación proveniente del host hacia él.
- Responder a peticiones estándar.- Cuando un dispositivo es alimentado, éste debe responder a peticiones estándar enviadas por el host durante la “enumeración”. El host podría también enviar peticiones en cualquier instante después de que se haya completado el proceso de enumeración. Todos los dispositivos deben responder a estas peticiones, las cuales solicitan las

capacidades y estatus del dispositivo. Al recibir una petición, el dispositivo coloca los datos o la información de estatus en un buffer para enviarlo al host.

- Checar errores.- Al igual que el host, un dispositivo agrega bits de chequeo de errores a los datos enviados. En la recepción de datos, los cuales incluyen bits de chequeo de errores, el dispositivo realiza los cálculos para verificar algún error en la transmisión. Con la respuesta del dispositivo o la ausencia de ésta le informa al host si se requiere re-transmitir los datos. El hardware del dispositivo normalmente realiza estas funciones, por lo que el desarrollador del firmware normalmente no se preocupa de estas funciones.
- Manejo de Potencia.- Un dispositivo podría tener su propia fuente de alimentación, obtener alimentación del bus USB o usar ambas fuentes de alimentación. Un host puede solicitar al dispositivo que entre en modo de baja potencia, estado de suspensión, lo cual significa que el dispositivo no debe consumir más de 2.5mA de la corriente del bus.

3.2.4 Desarrollando un dispositivo

El diseño de un dispositivo USB para PCs implica que el dispositivo entre en funcionamiento y el desarrollo de software requerido para comunicarse con él (driver).

Un dispositivo USB requiere lo siguiente:

- Un chip controlador del dispositivo con una interfaz USB y un CPU que se comunique con el controlador. El CPU puede estar en el mismo chip que en el controlador o en un diferente chip. Para nuestro caso tenemos un DSP 5509A, el cual contiene tanto al CPU como al controlador USB integrado.
- Un código de programa, hardware, o una combinación de estos para cumplir con las comunicaciones USB en el dispositivo.
- Hardware y código para realizar funciones específicas del dispositivo como son: lectura desde las entradas, escritura hacia las salidas, procesamiento de datos, etc. En nuestro caso además de llevar acabo la comunicación USB el dispositivo, realiza adquisición de datos analógicos, procesamiento de los datos adquiridos y manejo de salidas digitales para información de estatus de la aplicación.

El Host que se comunica con el dispositivo requiere lo siguiente:

- Hardware y software para el controlador del Host, los cuales típicamente están incluidos con el sistema operativo.
- Controlador del dispositivo (driver) instalado en el host, el cual permite que las aplicaciones desarrolladas en el host se puedan comunicar con el dispositivo. El driver podría estar incluido con el sistema operativo o ser provisto por el fabricante del dispositivo, por la compañía del chip utilizado en el dispositivo o por alguna otra fuente. En nuestro caso utilizamos un driver de uso libre llamado “libusb”.
- El software para la aplicación en el host, permite a los usuarios acceder al dispositivo. Para dispositivos estándar como son: mouse, teclado o disco duro no se requiere ningún software de aplicación, sin embargo para un dispositivo no genérico, como es en nuestro caso, requerimos la realización de una aplicación en el host.

3.2.4.1 HERRAMIENTAS PARA DESARROLLO

Para desarrollar un dispositivo USB, se requiere de las siguientes herramientas:

- Un compilador para crear el firmware del dispositivo, el código que corre dentro del chip del dispositivo. Para este trabajo utilizamos la herramienta IDE del Texas Instruments Code Composer Studio 3.0.
- Un dispositivo para programar el chip, el cual nos permite introducir el código compilado dentro de la memoria de programa del controlador. Para este trabajo utilizamos el emulador XDS510 USB-JTAG de Spectrum Digital.
- Un compilador para escribir y depurar software en el host, el cual podría incluir una combinación del controlador del dispositivo y del código de la aplicación. Para este trabajo utilizamos un controlador libre y para el desarrollo del código de la aplicación utilizamos LabWindows/CVI.
- También se recomienda utilizar un programa monitor para depurar el firmware del dispositivo y un analizador del protocolo para revisar el tráfico por el puerto USB. Para este trabajo se utiliza un osciloscopio digital como un analizador del protocolo.

3.2.4.2 PASOS PARA DESARROLLAR UN PROYECTO.

Los pasos para desarrollar un proyecto incluye las decisiones iniciales, la enumeración y el intercambio de datos [46].

- Decisiones Iniciales.- Antes de empezar a programar, se necesita seleccionar tanto el hardware para el dispositivo y el manejador (driver) en el host. Se requiere definir los requerimientos específicos del dispositivo, como puede ser: definir la taza requerida para la transferencia de datos y los requerimientos del tiempo o ancho de banda. Se necesita consideras todas las funciones que requiere desarrollar o realizar el dispositivo, por ejemplo lectura de entradas analógicas, tipo de procesamiento, almacenamiento alterno de datos, etc. Además se requiere que el desarrollador verifique si la PC podrá accesar al dispositivo utilizando un driver incluido en el sistema operativo o un driver customizado. Y por supuesto se requiere realizar una selección del chip que maneje la comunicación USB.
- Enumeración.- Para que el host (PC) sea capaz de realizar la enumeración en el dispositivo a desarrollar se requiere tomar en cuenta los siguientes puntos:
    1. Escribir el firmware del dispositivo para responder a peticiones USB estándar desde el Host y a otros eventos sobre el bus. En el proceso de enumeración el Host solicita al dispositivo una serie de descriptores, los cuales son estructuras de datos que describen las capacidades del dispositivo USB.
    2. Para un Host con sistema operativo Windows, se requiere identificar o crear un driver del dispositivo y un archivo INF para habilitar la identificación y comunicación con el dispositivo. El archivo INF es un archivo de texto que se encarga de nombrar al driver del dispositivo que se utilizará en el Host. Si el dispositivo empata con un clase soportada por Windows, entonces es posible utilizar un archivo INF incluido en Windows. Por lo pronto sólo se trabaja bajo esta plataforma de sistema operativo.
    3. Construir o adquirir una tarjeta de desarrollo, con la cual se pueda verificar el chip y la funcionalidad del firmware desarrollado.

4. Finalmente solo resta cargar el código dentro del chip y conectar el dispositivo al bus USB de la PC. Así un Host con Windows intentará enumerar al dispositivo y añadirlo al manejador de dispositivos. Una vez que esto ocurra, el Host ha establecido una comunicación con el dispositivo.

- Intercambio de Datos.- Una vez que el dispositivo ha sido enumerado correctamente, se continuará agregando componentes y código hasta completar la funcionalidad completa deseada para el dispositivo a desarrollar. Si se requiere, es momento de desarrollar una aplicación en el Host para comunicarse y probar el dispositivo. Cuando el código tanto del firmware en el dispositivo como el de aplicación en el Host hayan sido verificados, se recomienda entonces proceder a realizar el hardware final.

3.2.5 Transferencias de la interfaz.

Para desarrollar una aplicación que interactúe con un dispositivo USB no es necesario entender a detalle lo que pasa dentro de una transferencia en el bus USB. Pero si es importante entender cierta información sobre cómo se realizan las transferencias para poder decidir qué tipo de transferencia es mejor utilizar, para desarrollar el firmware y verificación del dispositivo.

Para enviar o recibir datos, un Host debe iniciar la transferencia por USB. Cada transferencia utiliza un formato definido por el estándar de USB para el envío de datos, para el direccionamiento de la información, con bits de detección de errores e información de estatus y control. Este formato varía con el tipo de transferencia y la dirección de la información.

Una comunicación USB se divide en dos categorías, dependiendo de cómo están interactuando el dispositivo con Host.

- Enlace de Configuración.- Aquí la comunicación es exclusivamente para que el Host aprenda del Periférico y ambos se preparan para el intercambio de información. Durante la enumeración, el Host le hace una serie de peticiones al dispositivo el cual debe identificar y contestar cada uno de estas peticiones, para después tomar acción de cada uno de los pedidos. En la computadora el

Sistema Operativo se encarga de realizar el proceso de la enumeración sin que el desarrollador tome parte aún, en este proceso el sistema le asigna un controlador al dispositivo mediante el archivo INF y entonces ya se puede dar la siguiente etapa.

- Enlace de Aplicación.- Aquí la comunicación está dedicada a la transacción de información entre la aplicación desarrollada en el Host y el dispositivo para lo cual fue destinado éste. Para acceder a los datos que se recogen del dispositivo se tiene que acceder a las aplicaciones de hardware USB con que cuente el sistema operativo tales como las funciones API del sistema operativo, y el firmware desarrollado en el dispositivo.

Pero para desarrollar un dispositivo USB si es necesario entender a detalle el proceso completo de una transferencia. Una transferencia está compuesta por transacciones y cada transacción por paquetes y cada paquete contiene información. Para entender cada uno de ellos debemos empezar por entender acerca de los “Puntos terminales” o “endpoints” y los “túneles” o “pipes”.

Puntos terminales o Endpoints

Un Endpoint es un buffer que almacena múltiples bytes, típicamente es un bloque de memoria de datos o un registro en un microcontrolador. Todo el tráfico en el bus USB parte desde o hacia un endpoint del dispositivo. Los datos almacenados en un endpoint podrían ser datos recibidos o datos que esperan ser transmitidos. El Host también tiene buffers en donde se almacenan los datos recibidos o los datos listos para ser transmitidos, pero no son identificados como endpoints. Así pues un endpoint sólo está definido en un dispositivo USB y no en el Host. Por definición un punto terminal lleva información en un sólo sentido, siempre y cuando no se trate de una transferencia de control, lo cual se verá más adelante.

Hay que tener especial atención con el direccionamiento de los endpoints, ya que estos son definidos únicamente en el dispositivo pero su dirección la da el Host. Esto es, los endpoints tienen como características un número específico del propio endpoint que varía de 0 a 15 y el sentido del flujo de información o dirección el cual se define desde la perspectiva del host:

- Entrada (IN): Los datos fluyen del dispositivo hacia el host.
- Salida (OUT): Los datos fluyen del host al dispositivo.

Sólo en una transferencia de control se tiene un par compuesto de puntos terminales IN y OUT que comparten un sólo número de punto terminal, que por lo general es el endpoint 0.

Todos los dispositivos USB deben tener un endpoint cero configurado como un endpoint de control y en raras ocasiones se requerirá usar otra transferencia de control las cuales sólo las soportan algunos microcontroladores, para nuestra aplicación, el dispositivo que se maneja si soporta la creación de otro endpoint de control.

Para las otras transferencias diferentes a la de control, la información fluye en una sola dirección, sin embargo en un flujo de estatus y control los datos pueden ir en direcciones opuestas. Un sólo número de endpoint puede soportar una transferencia IN y una OUT configurando debidamente el dispositivo.

Cada transacción en el bus de USB inicia con un paquete de datos que contiene el número del endpoint, un código que indica la dirección del flujo de datos y si la transacción está iniciando una transferencia de control. Lo cual se puede apreciar mejor en la siguiente tabla:

Tipo de Transacción	Fuente de Información	Transferencias que lo usan	Contenido
Entrada (IN)	Dispositivo	Todas	Datos o información de estatus
Salida (OUT)	Host	Todas	Datos o información de estatus
Configuración (SETUP)	Host	Control	Una petición

Tabla 3.2 Características de endpoints.

En cada transacción sobre el bus USB, el host envía un direccionamiento triple, el cual consiste en: una dirección del dispositivo, un número de endpoint y una dirección del endpoint. Al recibirse un paquete de salida (out) o de configuración (setup), el endpoint almacena los datos siguientes del paquete y típicamente el hardware del dispositivo

dispara una interrupción. Así entonces el firmware del dispositivo podrá procesar los datos recibidos y ejecutar alguna otra acción requerida en el dispositivo.

Al recibir un paquete de entrada (in), el endpoint deberá tener datos listos para ser enviados al host y proceder a enviarlos al bus, y por ultimo típicamente se dispara una interrupción para que el firmware pueda realizar lo que fuese necesario para que tenga listos los datos a enviar en la siguiente transacción de entrada. Cuando un endpoint no tiene datos listos para enviar o recibir en respuesta a paquetes de entrada o salida, entonces el endpoint envía un código de estado.

Túneles o Pipes

Antes que se dé una transferencia de información entre el Host y el dispositivo se debe establecer un túnel o camino virtual conocido como pipe. Un pipe es una asociación entre un endpoint del dispositivo y el software del controlador en el Host. El software del host es el encargado de establecer un pipe con cada dirección de endpoint al cual el Host quiera comunicarse.

El Host establecerá los pipes durante el proceso de enumeración. Si el usuario desconecta el dispositivo del bus, entonces el Host remueve los pipes que ya no son necesarios. Por otro lado, el Host también podrá solicitar la creación de nuevos pipes o la destrucción de pipes que ya no sean necesarios, utilizando transferencias de control para solicitar una configuración o interfaz alternativa para un dispositivo. Por default, cada dispositivo tiene un pipe de control, el cual utiliza al endpoint cero.

La información de configuración recibida por el Host incluye un descriptor o estructura del endpoint, por cada endpoint que se requiera utilizar. Cada descriptor de un endpoint contiene: la dirección del endpoint, el tipo de transferencia que soporta, el tamaño máximo de paquetes de datos, y en algún caso, el intervalo deseado para las transferencias.

Debido a que el USB fue diseñado para manejar muchos tipos de periféricos con requerimientos diferentes ya sea en su tasa de transferencia, tiempo de respuesta y corrección de errores, para ello se cuenta con 4 tipos de transferencia de datos, cada una para diferentes necesidades, la tabla 5.3 nos resume cada una de ellas.

Cada dispositivo puede soportar cualquiera de los 4 tipos de transferencias y deberá usar las más adecuadas para su propósito. Todos los dispositivos, por especificación, deberán utilizar por lo menos la transferencia de control.

Tipo de Transferencia	Control	Avalancha **(**Bulk**)**	Interrupción (Interrupt)	Isócrona (Isochronous)
Uso típico	Identificación y configuración	Impresora, Scanner	Ratón, teclado	Audio, video
¿Requerida?	Sí	No	No	No
¿Permitida en dispositivos de baja velocidad?	Sí	No	Sí	No
Máximo tamaño de paquete en bytes (high speed)	64	512	1024	1024
Vel. Máxima garantizada en paquetes/intervalo (high speed)	Ninguna	Ninguna	3/125 µs	3/125 µs
Máximo tamaño de paquete en bytes (full speed)	64	64	64	1023
Vel. Máxima garantizada en paquetes/intervalo (full speed)	Ninguna	Ninguna	1/1 ms	1/1 ms
Máximo tamaño de paquete en bytes (low speed)	8	No permitida	8	No permitida
Vel. Máxima garantizada en paquetes/intervalo (low speed)	Ninguna	No permitida	1/10 ms	No permitida
Dirección del flujo de datos	IN y OUT	IN o OUT	IN, o OUT (para USB 1.0 sólo IN)	IN o OUT
Ancho de banda reservado para todas las transferencia de un tipo	10% para low/full speed, 20% para high speed	Ninguno	90% para low/full speed, 80% para high speed (máximo combinando iterrupción e isócrona)	
¿Transferencia de datos por mensaje o ráfaga (stream)?	Mensaje	Ráfaga	Ráfaga	Ráfaga
¿Corrección de errores?	Sí	Sí	Sí	No
¿Taza de transferencia garantizada?	No	No	No	Sí
¿Latencia garantizada?	No	No	Sí	Sí

Tabla 3.3 Características de las diferentes transferencias.

En la figura 3.16, se muestra un diagrama donde se desglosa las etapas de una transferencia. Una transferencia puede llevarse a cabo en multiple frames o microframes.

Token → Identificación

Handshake → Reconocimiento

PID → Identificador de paquete

CRC → Chequeo de errores (Comprobación de Redundancia Cíclica)

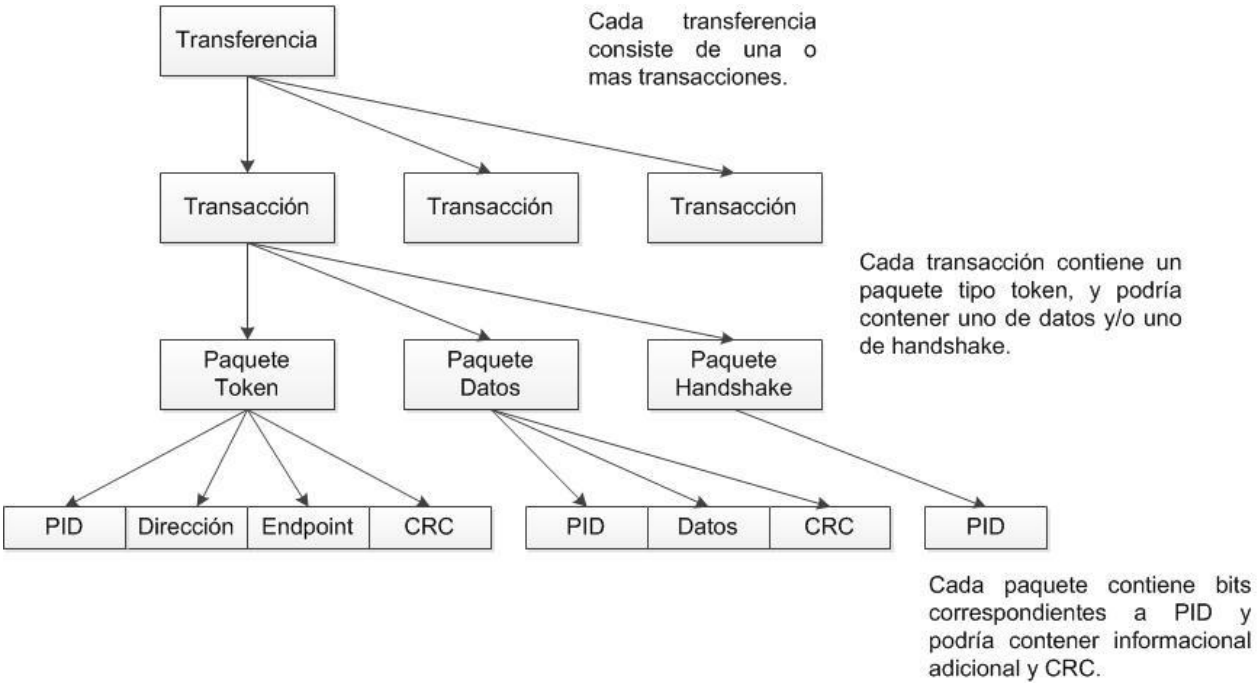


Figura 3.16 Desglose de una transferencia USB.

3.2.5.1 TRANSFERENCIA TIPO CONTROL

Como ya se ha mencionado, este tipo de transferencia deberá estar presente en cualquier dispositivo USB, ya que mediante esta transferencia se llevan a cabo peticiones desde el Host para poder saber que dispositivo se está conectando a él y para configurar correctamente cualquier dispositivo USB.

Al ser una transferencia soportada para todos los dispositivos, siempre se asigna el endpoint cero como el que ejecuta esta transferencia y no hay necesidad de asignar más endpoints para esta transmisión.

Para estas transferencias se manejan tres estados conocidos como Configuración (Setup), Datos (Data) y Estatus (Status), donde cada estado consiste de una o más transacciones. Sabiendo que las transferencias de control requieren transferir información en ambas direcciones (del host al dispositivo y viceversa) se configura las direcciones del endpoint tanto de entrada (IN) como el de salida (OUT).

En la figura 3.17 se puede apreciar un ejemplo de una transferencia de escritura en un endpoint de control y en la figura 3.18 se aprecia la correspondiente transferencia pero para una de lectura. Estos diagramas aplican para transacciones usadas con dispositivos low and full speed sobre puertos high speed. Se considera estos ejemplos ya que la aplicación actual está desarrollada de esta manera, el dispositivo es un full speed y la mayor cantidad de puertos usb en las PCs actuales son tipo high speed.

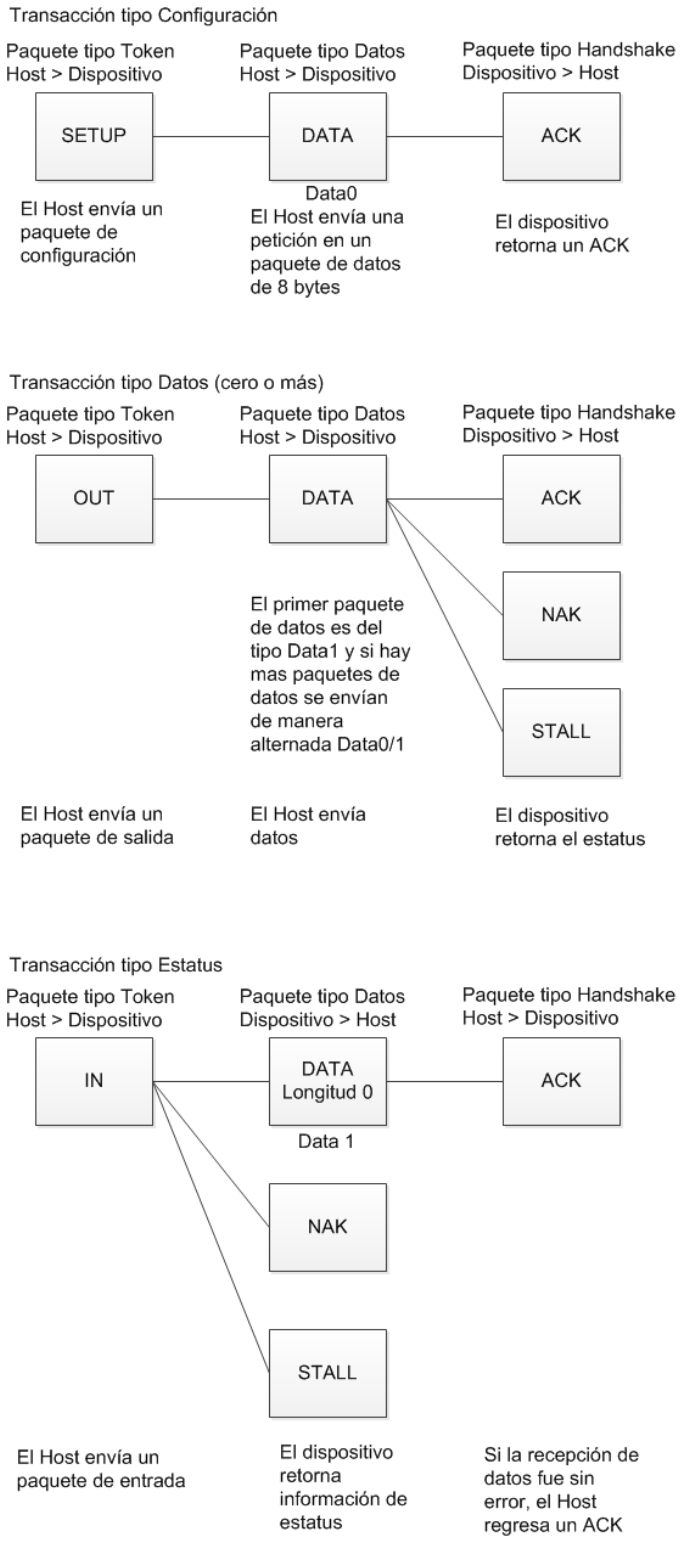


Figura 3.17 Transferencia de control tipo Escritura.

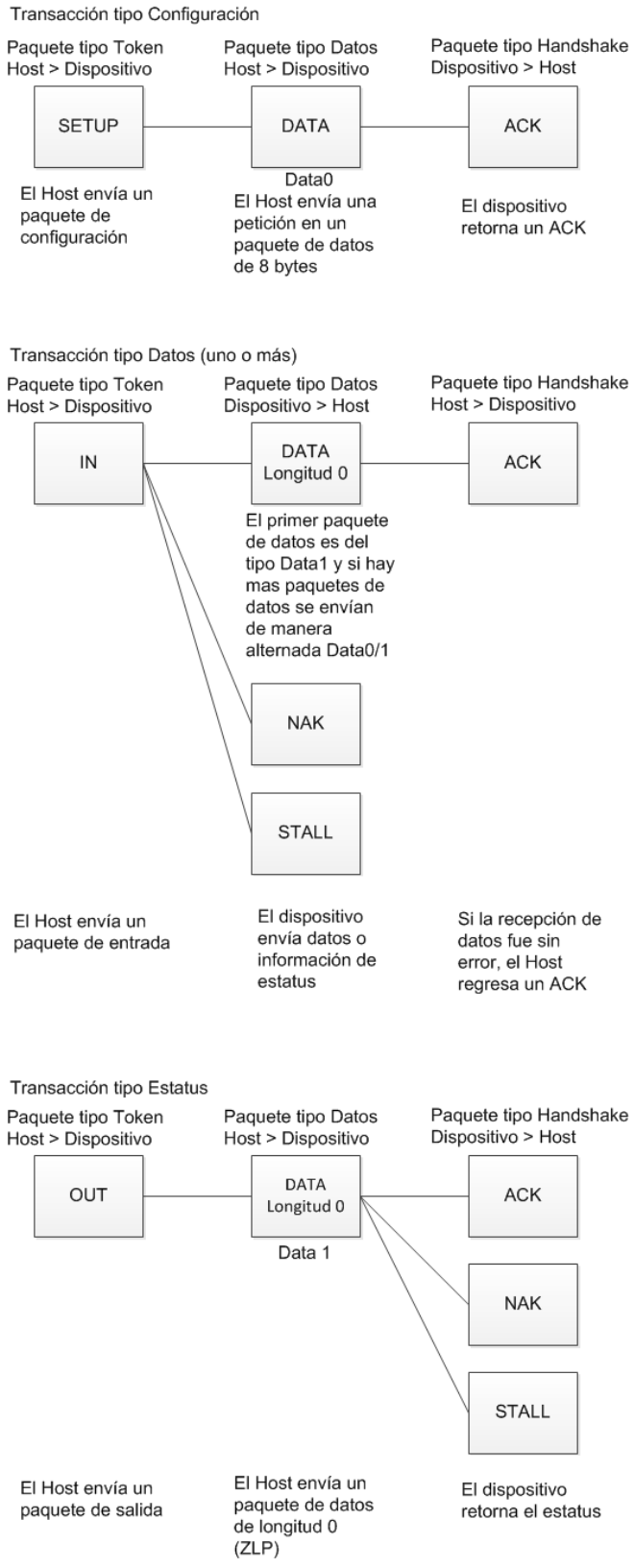


Figura 3.18 Transferencia de control tipo Lectura.

El tamaño de datos que se puede enviar en una transferencia de control depende del tipo de bus utilizado, para un bus “low speed” se maneja un paquete de datos máximo de 8 bytes; para un bus “full speed” el tamaño máximo del paquete de datos es de 8, 16, 32 o 64 bytes; para un bus “high speed” se tiene un tamaño máximo de 64 bytes y finalmente para el “super speed” será de 512 bytes.

Estos bytes especificados son únicamente para el paquete de datos no se incluye el tamaño de los paquetes PID y CRC. En el estado de Datos, todos los paquetes deben de ser del tamaño máximo para el endpoint en uso excepto el último paquete. El tamaño máximo del paquete es especificado en el descriptor del dispositivo.

Si una transferencia contiene más datos de los que soporta una transacción, entonces los datos son enviados o recibidos por el host en muchas transacciones.

Para algunas transferencias de lectura, la cantidad de datos enviados del dispositivo hacia el host puede variar, el dispositivo deberá indicarle al host cuando ya no existan más datos a enviar, y esto se logra retornado un paquete de datos de longitud cero (ZLP) en respuesta a una petición de datos desde el host.

En cuanto a la velocidad de las transferencias, es responsabilidad del host asegurar que todas las transferencias de control se completen lo más rápido posible. Para esto el controlador del host reserva una porción del ancho de banda del bus para este tipo de transferencias. Para el caso de buses “low” y “full speed” se reserva un 10% para las otras versiones se reserva un 20%. Pero si las transferencias no requieren este ancho de banda entonces el remanente se podrá usar para transferencias tipo “bulk”.

Cuando exista una petición no soportada en el dispositivo, éste deberá enviar al host un paquete STALL en el estado de datos o de estatus.

3.2.5.2 TRANSFERENCIA TIPO BULK

Transferencias tipo Bulk son útiles para transferir datos cuando el tiempo no es crítico. Estas transferencias se caracterizan por enviar grandes cantidades de datos sin saturar el bus USB, debido a que cada transferencia se espera un tiempo hasta que haya disponibilidad en el bus para enviar los siguientes datos [47].

Principalmente estas transferencias se utilizan en el envío de datos a una impresora o un scanner y para leer o escribir a una unidad de memoria o disco duro. Si el bus se encuentra libre, este tipo de transferencia es la más rápida.

Dispositivos “low speed” no soportan este tipo de transferencias. No se requiere que este tipo de transferencia se implementada en un dispositivo, a diferencia de las transferencias de control, en donde si es requerida dicha transferencia. Más sin embargo, si el dispositivo es de alguna clase en especial podría requerir esta transferencia. Por ejemplo, un dispositivo para almacenamiento de datos debe tener un endpoint configurado para transferencias bulk, tanto para entrada como para salida.

Una transferencia bulk, consiste de una o más transacciones tipo IN u OUT. Transferencias bulk e interrrupt tienen la misma estructura pero diferentes tiempos, figura 3.19. Todos los datos viajan en una sola dirección, para transferir datos en ambas direcciones se requiere configurar un túnel (pipe) y endpoint diferente para cada dirección.

Una transferencia bulk, termina correctamente cuando la cantidad esperada de datos se ha transferido completamente. La especificación USB por sí misma no define un protocolo para indicar el número de bytes de datos para una transferencia de este tipo. Si se requiere esta información, entonces el dispositivo y el host podrán utilizar un protocolo definido por el fabricante del chip o uno específico de una clase.

El tamaño de datos que se puede enviar en una transferencia bulk depende del tipo de bus utilizado, para un bus “full speed” el tamaño máximo del paquete de datos es de 8, 16, 32 o 64 bytes; para un bus “high speed” se tiene un tamaño máximo de 512 bytes y finalmente para el “super speed” será de 1024 bytes.

Estos bytes especificados son únicamente para el paquete de datos no se incluye el tamaño de los bits correspondiente a PID y CRC. Durante la enumeración el host lee el tamaño máximo del paquete para cada endpoint tipo bulk desde los descriptores del dispositivo. Si los datos a enviar no caben dentro de un solo paquete, entonces el host usa múltiples transacciones hasta completar la transferencia.

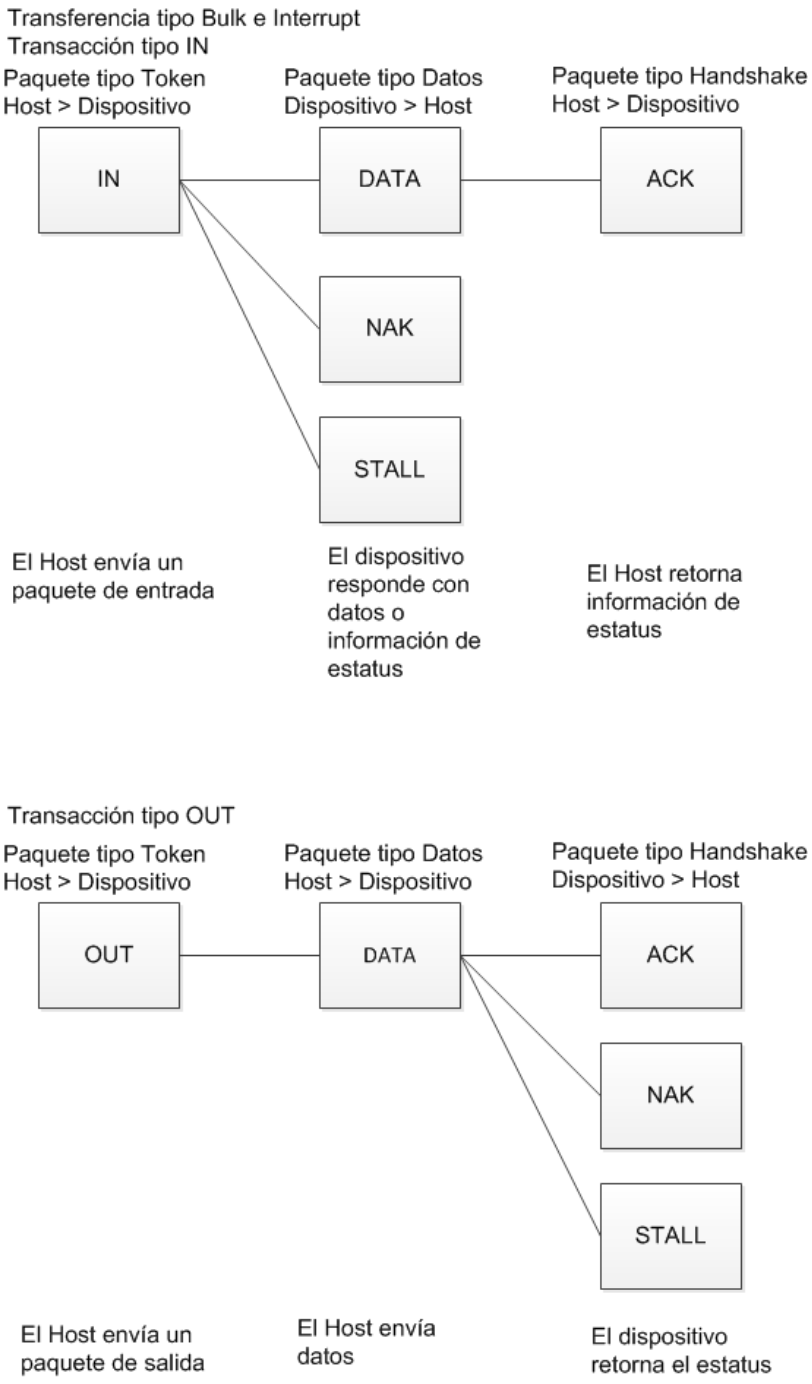


Figura 3.19 Transferencia tipo Bulk e Interrupt.

A diferencia de las transferencias de control, en las transferencias bulk no se reserva un ancho de banda específico, por lo que si el bus está muy ocupado sobre con otras transferencias, el tiempo de la transferencia bulk podría ser muy largo. Lo que si sucede en estas transferencias es que el host garantiza éstas se completarán eventualmente.

Sin embargo, cuando el bus esta sin ocuparse, las transferencias bulk pueden usar la mayor parte del ancho de banda de éste, y así tener un overhead bajo y ser la transferencia más rápida.

Es importante recalcar que en el mundo real el rendimiento varía con la arquitectura del host y con el hardware del controlador en el host y su driver, incluyendo latencias por el acceso a memoria del sistema.

3.2.5.3 TRANSFERENCIA TIPO INTERRUPT

Transferencias Interrupt son útiles cuando los datos se tienen que transferir sin ningún retardo. Aplicaciones típicas incluyen teclados, controladores de juego entre otros. Por ejemplo el usuario no querrá notar un retardo entre presionar una tecla en el teclado y ver el resultado en pantalla. Dispositivos de baja velocidad “low speed”, son los que principalmente utilizan este tipo de transferencias.

En buses de velocidades low y full, el ancho de banda disponible para esta transferencia es limitado, pero en buses High Speed y SuperSpeed no existen estos límites.

Las transferencias tipo interrupt son más parecidas a una interrupción, ya que éstas garantizan una respuesta rápida desde el host, y al igual que los endpoints tipo bulk típicamente utilizan interrupciones en el firmware para detectar nuevos datos recibidos. Al igual que transferencias bulk, las interrupt deberán esperar que el host solicite los datos antes de iniciar las transferencias.

Todas los tipos de dispositivos y buses permiten este tipo de transferencias, aunque no es requerido que todos los dispositivos configuren endpoints de este tipo.

Una transferencia interrupt consiste de una o más transacciones de entrada o salida. Para transferir datos en ambas direcciones se requiere que se configure un túnel (pipe) y endpoint diferente para cada dirección. Como se puede apreciar en la figura 3.19 la estructura de estas transferencias es idéntica a la tipo bulk con la única diferencia que la transferencia interrupt garantiza una latencia entre cada transacción.

Una transferencia tipo Interrupt, termina satisfactoriamente cuando la cantidad esperada de datos se hayan transferido. Al igual que transferencias bulk, la especificación de USB no define un protocolo para especificar la cantidad de datos en una transferencia. Cuando se requiere especificar, entonces el dispositivo y el host podrán utilizar algún protocolo definido por la clase o por el fabricante del chip para pasar esta información.

Este tipo de transferencias garantizan una latencia máxima, o tiempo entre intentos de transacción. En otras palabras, no existe una razón de transferencia garantizada, sólo la garantía que el host tendrá un ancho de banda disponible para un intento de transacción en cada periodo de latencia.

El descriptor de un endpoint de este tipo, especifica el máximo periodo de latencia. Para dispositivos de baja velocidad (low speed), la máxima latencia puede ser desde 10 hasta 255 ms. Para Full speed, el rango es de 1-255 ms. Para High speed el rango es de 125 µs hasta 4.096 s en incrementos de 125 µs.

El host puede iniciar cada transacción en cualquier tiempo dentro de la latencia máxima especificada desde que inicio la transacción previa. Por ejemplo, sobre un bus full speed con una latencia máxima de 10 ms, 5 transferencias pudieran ocurrir en un tiempo de 50 ms o de 5 ms dependiendo de la duración de cada transacción.

3.2.5.4 TRANSFERENCIA TIPO ISOCHRONOUS

Transferencias Isochronous son tipo streaming o también llamadas transferencias en tiempo real, las cuales son útiles cuando los datos deben llegar en frecuencia constante o con un tiempo límite específico y donde errores ocasionales son tolerables. Para full speed y high speed.

Las transferencias isochronous pueden transferir más datos por frame o intervalo del bus que las transferencias interrupt, pero las isochronous no soportan retransmisión de datos automáticamente al ocurrir un error de datos, figura 3.20.

Una transferencia isochronous es una manera de asegurar que para cualquier bloque de dato se ha reservado un ancho de banda en un bus ocupado. A diferencia de las

transferencias bulk, el host garantiza que una configuración solicitada del ancho de banda estará disponible. Dispositivos low speed no soportan estas transferencias.

Isochronous significa que los datos tienen una frecuencia de transferencia constante, con un número definido de bytes transferidos en cada frame o intervalo del bus. Una transferencia de este tipo consiste de una o más transacciones de entrada o salida en intervalos iguales.

Al igual que en las transferencias bulk e interrrupt, para transferir datos en ambas direcciones se requiere configurar un endpoint y pipe diferente para cada dirección. Además que el estándar no especifica un protocolo para definir la cantidad de datos a transmitir.

El controlador del host es el encargado de determinar si existe el ancho de banda suficiente para reservar una transacción tipo Isochronous. Cada dispositivo que tenga configurado un endpoint isochronous debe tener un endpoint diferente para que el host pueda configurar adecuadamente al dispositivo.

Para tener un ancho de banda óptimo, un dispositivo de este tipo podría tener un endpoint alterno con un paquete de datos menor. Logrando con esto que el driver del dispositivo pueda entonces solicitar el uso del endpoint que transfiera datos en una frecuencia más lenta, si fuese necesario. Una vez que el host configure al dispositivo y se seleccione una interfaz, las transferencias son garantizadas para tener el tiempo requerido.

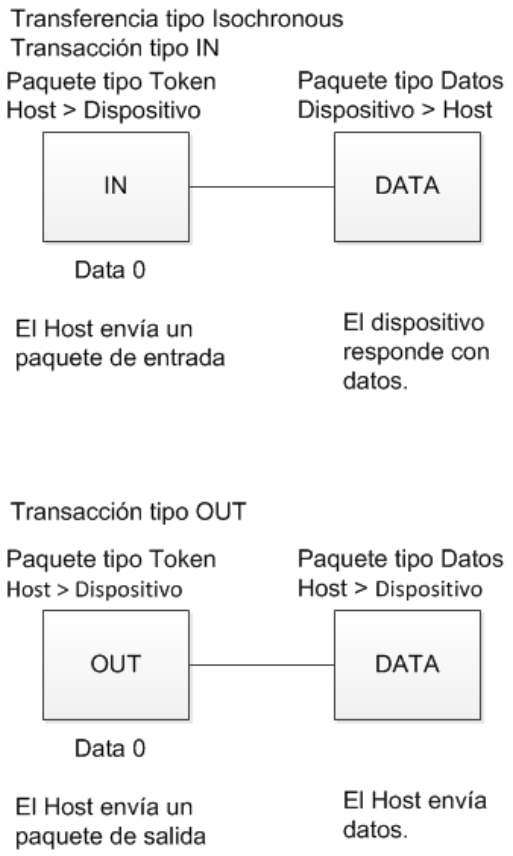


Figura 3.20 Transferencia tipo Isochronous.

3.2.6 Enumeración.

Enumeración es el proceso donde el host aprende acerca del dispositivo, identifica que dispositivo se quiere conectar. Aquí se da el primer intercambio de información entre ambas partes y éste proceso se debe realizar lo siguiente:

- Asignar una dirección al dispositivo.
- Leer la estructura de datos del dispositivo.
- Asignar y cargar el controlador respectivo.
- De las opciones presentadas en la información recuperada, seleccionar la configuración en que trabajará en adelante.

Después de este proceso el dispositivo ya está configurado y listo para transferir información con los puntos terminales (endpoints) debidamente configurados.

Basándonos en la especificación de USB 1.1, esta define seis estados del dispositivo. Durante el proceso de enumeración el dispositivo pasa a través de estos cuatro

estados: Energizado, Default, Direccionado y Configurado. Los otros dos que menciona la especificación son: Añadido y Suspendido. En cada estado el dispositivo tiene definidas ciertas capacidades y funcionamiento.

3.2.6.1 PASOS EN EL PROCESO DE ENUMERACIÓN

A continuación se mencionan los pasos que sigue el proceso de enumeración, los cuales son una secuencia típica de eventos que ocurren en este proceso. El firmware del dispositivo no debería asumir que las peticiones de enumeración y los eventos ocurrirán en un orden en particular. Para funcionar satisfactoriamente, un dispositivo debe detectar y responder a cualquier petición de transferencia de control o algún otro evento del bus en cualquier momento.

1. **El usuario conecta físicamente el dispositivo a un puerto USB.** También se da cuando el sistema operativo inicia con el dispositivo ya conectado. El puerto puede estar en el hub raíz o en uno conectado en cascada, aquí es donde el dispositivo se energiza y queda en estado de Energizado.
2. **El hub detecta al dispositivo.** El hub monitorea el voltaje en cada línea de sus puertos. El hub tiene resistencias pull-down de 15 Kohm en cada uno de sus puertos, mientras que el dispositivo tiene una resistencia de 1.5 Kohm pull-up en el puerto D+ para los dispositivos que trabajen a full-speed o en D- para los que trabajen en low-speed. Los dispositivos que trabajen en Hi-speed se conectan como los de full-speed.
3. **El host aprende del nuevo dispositivo.** Cada hub utiliza su endpoint de Interrupción para notificar si ocurren eventos en el propio hub. La notificación indica solamente si el hub o si algún puerto (y cual) ha experimentado un evento. Durante este paso el host le envía al hub una petición de GET_PORT_STATUS, esta petición es estándar ya que cualquier hub soporta dicha petición. De la información recopilada el host sabe cuándo un nuevo dispositivo es conectado.
4. **El hub detecta si un dispositivo es de baja o completa velocidad.** Justo antes de que el hub reinicia el dispositivo, el hub determina si el dispositivo es de velocidad baja o completa examinando los voltajes en las líneas D+ y D-. El hub puede detectar la velocidad del dispositivo, determinando cual línea tiene voltaje más alto cuando el dispositivo se encuentra en reposo. El hub envía la

información al host en respuesta de la siguiente petición de GET_PORT_STATUS. Un dispositivo USB 1.x permite que el hub detecte su velocidad justo después del reinicio. Las versiones de USB 2.0 requieren, en cambio, que la detección de velocidad se dé justo antes del reincio de manera que él pueda averiguar si el dispositivo es de alta velocidad.

5. **El hub reinicia el dispositivo.** Cuando el host aprende del nuevo dispositivo, el controlador del host le envía al hub una petición de SET_PORT_FEATURE, la cual le solicita al hub que reinicie el puerto. El hub coloca las líneas de datos del dispositivo en condición de Reset por al menos 10 milisegundos, esto se hace sólo cuando hay un nuevo dispositivo, mientras que otros hubs o dispositivos en el bus ignoran dicha señal de reset. Reset es una condición especial donde tanto D+ y D- tienen un nivel lógico bajo, normalmente las líneas de datos tienen niveles lógicos opuestos.
6. **El host aprende si un dispositivo de velocidad completa soporta alta velocidad.** Para este paso se emplean dos estados especiales de señales. El estado Chirp J en donde solamente se usa la línea D+, y el estado Chirp K para la línea D-. Durante el reinicio, un dispositivo de alta velocidad envía un estado Chirp K que lo detecta un hub, también de alta velocidad, y le responde con estados alternados de Chirp K y J, cuando el dispositivo detecta el patrón KJKJKJ, retira los pull-up de velocidad completa y establece una comunicación de alta velocidad para todas las comunicaciones futuras. Si por el contrario, el hub no responde al estado Chip K del dispositivo, entonces el dispositivo sabrá que debe continuar en modo velocidad completa.
7. **El hub establece una señal para abrir camino entre el dispositivo y el bus.** El host verifica que el dispositivo ha salido del estado de reinicio mediante una petición GET_PORT_STATUS. Un bit dentro de la información retornada se encarga de esto. Cuando el hub quita el estado de reinicio, el dispositivo queda en su estado por defecto, y sus registros USB están en estado de reinicio, entonces el dispositivo está listo para responder a transferencias de control usando el endpoint 0; el dispositivo ya puede comunicarse con el host usando la dirección por defecto 00h.

8. **El Host envía una petición GET_DESCRIPTOR para saber el tamaño máximo de paquete del túnel por defecto.** En vista que el host enumera un dispositivo a la vez, se usa siempre la dirección 0 y el endpoint 0. El octavo byte del descriptor del dispositivo contiene el tamaño máximo de paquete que va a soportar el endpoint 0. Un host de Windows solicita 64 bytes, pero después de recibir un paquete (tenga o no 64 bytes), empieza la etapa de status en la transferencia. Para completar el estado de estatus el sistema operativo le solicita al hub que reinicie al dispositivo como se mencionó previamente en el paso 5. De acuerdo a las especificaciones, no se es necesario el reinicio ya que los dispositivos deberían ser capaces de aceptar que el host abandone una transferencia de control en cualquier momento respondiendo al siguiente paquete de Setup, pero el reinicio es una precaución que asegura que el dispositivo estará en un estado conocido cuando termine el reinicio.
9. **El host asigna una dirección.** Cuando el reset es completado, el controlador del host asigna una única dirección al dispositivo enviándole una petición SET_ADDRESS. El dispositivo completa el estado de estatus utilizando la dirección por defecto y entonces desde ahí se asigna la nueva dirección. El dispositivo se encuentra ahora en el estado Direccionado. Todas las comunicaciones a partir de este punto usan la nueva dirección, esta nueva dirección es válida hasta que el dispositivo sea desconectado del bus, un hub reinicie el puerto, o que el sistema se reinicie completamente. En la siguiente enumeración, el host podría asignar una dirección diferente al dispositivo.
10. **El host aprende las habilidades del dispositivo.** El host envía una petición GET_DESCRIPTOR a la nueva dirección para leer el descriptor del dispositivo, esta vez el host recupera toda la información del descriptor. El descriptor es una estructura de datos que contiene el tamaño máximo del paquete para el punto terminal 0, el número de configuraciones que el dispositivo soporta e información básica adicional del dispositivo. El host solicita uno o más descriptores de configuración, los cuales están especificados en el descriptor del dispositivo.
11. **El host asigna y carga el controlador del dispositivo.** Después de aprender todo lo que se pueda del dispositivo a partir de sus descriptores, el host empieza la búsqueda del controlador que mejor le cuadre para la administración de la

comunicación con el dispositivo. Hosts con sistema operativo Windows usan archivos INF para identificar controladores de dispositivos.

12. **El controlador del host selecciona una configuración.** En esta parte el controlador del host asignado solicita la configuración al dispositivo enviándole una petición SET_CONFIGURATION con el número de configuración deseada. La mayoría de dispositivos soportan un único tipo de configuración, pero si el dispositivo soporta múltiples configuraciones entonces el controlador podrá decidir cuál configuración usar en base a la información recogida del dispositivo o el driver le pide al usuario que elija. El dispositivo ahora se encuentra en el estado Configurado y la interfaz o interfaces del dispositivo están ya habilitadas.

A partir de este momento el dispositivo ya se encuentra listo para ser usado. Además de los estados que se mencionan en estos pasos, existen otros dos estados para el dispositivo, como se mencionó anteriormente, los cuales son: Añadido y Suspendido los cuales se pueden dar en cualquier momento.

- **AÑADIDO.** Si el hub no está proveyendo alimentación al dispositivo por su línea V_{BUS}, entonces el dispositivo se encuentra en este estado. La ausencia de potencia podría ocurrir si el hub ha detectado una condición de sobre-corriente o si el host le solicita al hub remover la alimentación del puerto. Sin voltaje en V_{BUS}, el host y el dispositivo no podrán comunicarse.
- **SUSPENDIDO.** Un dispositivo entra en este estado después de que no se detecte actividad en el bus, incluyendo marcadores de inicio de trama (SOF) por al menos 3 ms. En estado suspendido, el dispositivo debería limitar su uso de la alimentación del bus. Tanto dispositivos configurados como los no-configurados deben soportar este estado

3.2.7 Descriptores

Los descriptores son estructuras de datos que permiten al host conocer información del dispositivo. Cada descriptor contiene información diversa del dispositivo.

bTipo de Descriptor	Tipo de descriptor	¿Requerido?
01h	Dispositivo	Sí
02h	Configuración	Sí

03h	Texto	No, al menos que el driver lo requiera. Es un texto descriptivo opcional.
04h	Interfaz	Sí
05h	Endpoint	Sí, al utilizar un endpoint diferente al cero.
06h	Calificador_dispositivo	Sí para dispositivos que soportan full y high speeds. No permitido para otros dispositivos.
07h	Otra_configuración_velocidad	Sí para dispositivos que soportan full y high speeds. No permitido para otros dispositivos.
09h	OTG	Sí para dispositivos On-The-Go
0Ah	Depuración	No
0Bh	Asociación_interfaz	Sí para dispositivos compuesto
0Ch	Seguridad	Si para dispositivos inalámbricos
0Dh	Llave	
0Eh	Tipo de encriptación	
0Fh	Almacenamiento objeto del dispositivo binario (BOS)	Sí para dispositivos inalámbricos
10h	Capacidad del dispositivo	
11h	Compañero del endpoint inalámbrico	Si para dispositivos inalámbricos

Tabla 3.4. Descriptores definidos en la especificación USB 2.0.

3.2.8 El controlador de funciones

El controlador del dispositivo es el que hace posible la comunicación entre los controladores de USB del sistema y las aplicaciones que quieran acceder al dispositivo y a su vez al hardware que éste maneje el cual puede ser una impresora, módem, teclado, etc. Estos dispositivos pueden ser periféricos estándar o diseños específicos para aplicaciones especiales. Algunos controladores son conocidos como "Controladores Clase" que manejan las comunicaciones de una variedad de dispositivos con funciones similares.

Una característica del controlador del dispositivo es el de aislar a las aplicaciones de tener que saber o manejar cualquier detalle acerca de las conexiones físicas, señales y protocolos que se requieran en la comunicación con el dispositivo. Una aplicación es el programa con el que finalmente el usuario interactúa y soporta el hardware que necesite usar, en nuestro caso un hardware USB. El controlador tiene la misión de hacer de traductor entre el código de nivel de aplicación y el código de nivel de

hardware. El código de nivel de aplicación usa funciones soportadas por el sistema operativo para comunicarse con el controlador del dispositivo; y el código de nivel de hardware maneja los protocolos necesarios para acceder al circuito que maneje el dispositivo el cual incluye el detectar las etapas de las señales status y el uso o no de las señales de control en tiempos apropiados.

El sistema operativo ya sea Windows o Linux incluye funciones de interfaz de aplicación para programadores o su sigla en inglés API, que habilitan a las aplicaciones para comunicarse con el controlador del dispositivo. Las aplicaciones escritas en lenguaje de alto nivel como C, C++, Delphi o Python pueden llamar a las funciones API. Tres de las funciones que el controlador puede soportar para la lectura y escritura de los dispositivos USB son: Lectura (ReadFile), escritura (WriteFile) y entrada/salida de control (DeviceIOControl).

Para elegir un controlador que se acomode a las necesidades del diseñador es necesario saber con qué opciones se cuenta, algunas veces el sistema operativo ya tiene los controladores cargados, en otras ocasiones el chip con el que se trabaje los provee, algunas otras veces se pueda conseguir de algún sitio en la web. Si no se tiene acceso al controlador, o éste no existe, entonces será necesario escribir un controlador personalizado para el dispositivo seleccionado, para lo cual se puede contar con una variedad de herramientas que faciliten y acorten esta tarea. Algunas veces hay más de una manera de conseguir el controlador lo que dependerá del costo, facilidad y rendimiento que se considere a la hora de la elección del camino a tomar.

- DISPOSITIVO ESTÁNDAR.- Muchos dispositivos periféricos muy usados pertenecen a esta clase tales como disqueteras, impresoras, teclados, etc. Los cuales ya están dentro de un lista y para ellos el sistema operativo incluye a los controladores clase.
- DISPOSITIVOS PERSONALIZADOS.- Algunos periféricos son dispositivos personalizados hechos para una aplicación específica tales como unidades de adquisición de datos, controladores de motores, instrumentos de prueba o la aplicación que desarrolla en este trabajo. Estos dispositivos pueden usar controladores personalizados o pueden ser acomodados de tal forma que cumplan con los requerimientos de un controlador clase. Por ejemplo, un

sistema de adquisición de datos puede usar tranquilamente un controlador clase HID (Dispositivo de Interfaz Humano) que explicaremos más adelante.

Dentro del sistema operativo hay dos modos de correr el código de un controlador: kernel y usuario.

Cada uno permite un diferente nivel de privilegios de acceder a la memoria u otros recursos del sistema. Todas las aplicaciones deben correr en modo usuario. La mayoría de los controladores, incluidos todos los controladores USB, corren en modo kernel, sin embargo un dispositivo USB puede tener también un controlador en modo usuario complementario.

Normalmente el modo usuario está más restringido el acceso a la memoria y recursos del sistema que éste considere protegidos lo que hace posible que el sistema operativo corra diferentes aplicaciones a la vez, en cambio el modo kernel, el código tiene acceso irrestricto a tales recursos y memoria del sistema como son la capacidad de ejecutar instrucciones de administración de memoria y control de acceso a puertos de entrada salida E/S.

En Windows 98 y Me y linux, las aplicaciones pueden acceder a los puertos E/S directamente siempre y cuando un controlador de bajo nivel no haya reservado tal puerto. En Windows NT y 2000, solamente el modo kernel puede acceder a los puertos E/S. La figura 3.21 muestra los componentes más comunes tanto en modo usuario como el modo kernel usados en una comunicación USB.

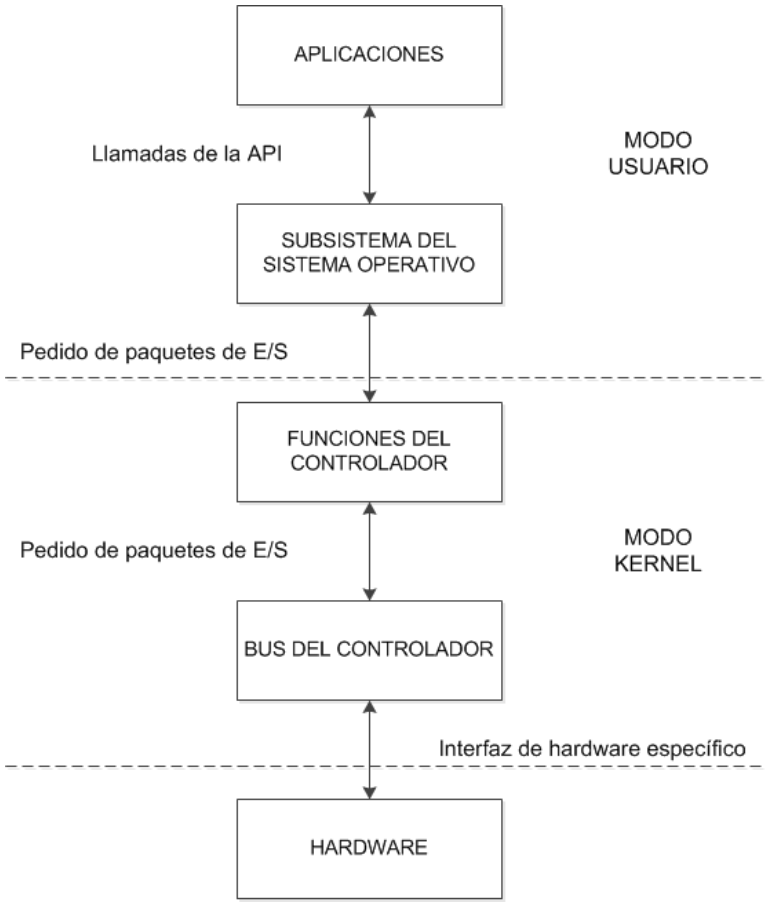


Figura 3.21 Capas de un modelo de controlador en el sistema operativo.

Un controlador de funciones le permite a la aplicación conversar con el dispositivo USB usando funciones API, las cuales son parte del subsistema Win32 en Windows y están a cargo de las funciones de usuario tales como correr aplicaciones, administrar las entradas del usuario a través del teclado o ratón y mostrar las salidas en el monitor, en Linux se encuentran en las librerías del directorio “usr”. El controlador de funciones sabe cómo comunicarse con los niveles más bajos del controlador del bus que maneja el hardware. La figura 3.22 muestra cómo trabajan tales controladores en una comunicación USB.

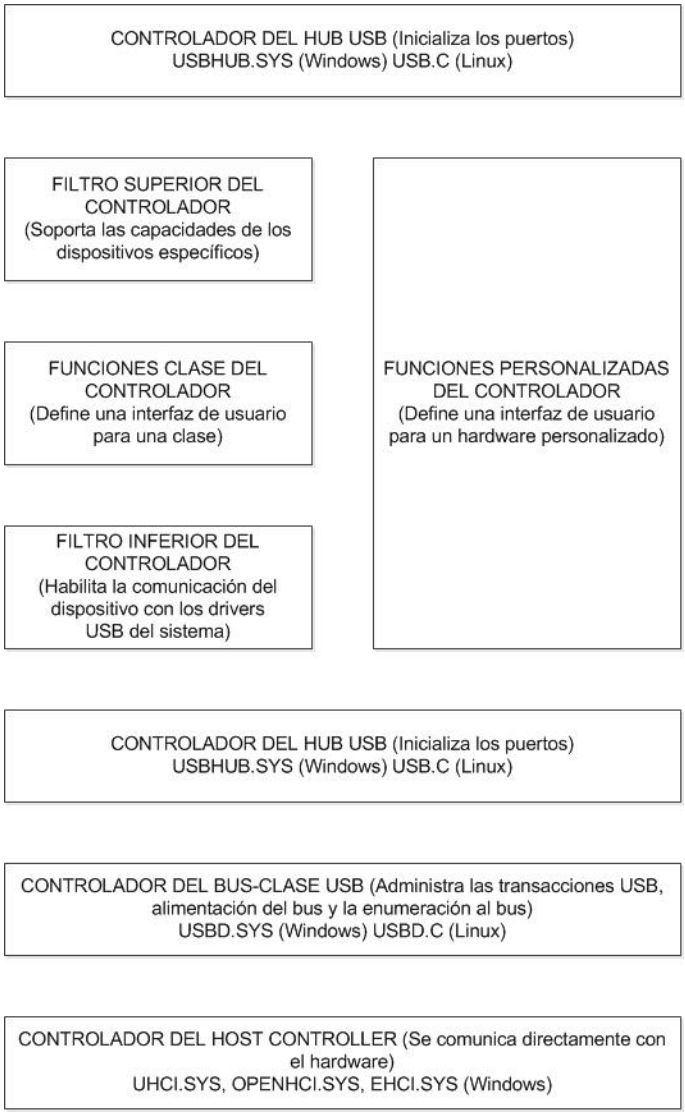


Figura 3.22 Comunicaciones realizadas por el controlador.

Existe el Host controller, controlador de la clase, controlador del hub y un controlador de funciones que puede tener más de un archivo. Normalmente se habla de un controlador del dispositivo, pero en el fondo éste no trabaja por sí sólo pues tiene que complementarse con el controlador de funciones y el controlador de bus para trabajar como uno solo.

Los dispositivos de interfaz humana o su sigla en inglés HID son los primeros que soporta el sistema operativo, por esta razón un dispositivo USB que pertenezca a esta clase son normalmente los más fáciles de implementar, siempre y cuando cumplan con las especificaciones que se requieran para pertenecer a esta clase.

La simple designación “interfaz humano” sugiere que el dispositivo deba interactuar directamente con las personas ya sea este un teclado, ratón, joystick, o algún otro de este tipo. Pero un dispositivo HID no tiene que ser necesariamente un interfaz humano, como se mencionó anteriormente sólo necesitamos que cumpla con las especificaciones dadas a continuación, lo que también lo restringe en algunas cosas.

El intercambio de datos se da en estructuras llamadas reportes. El firmware del dispositivo debe soportar el formato de un reporte HID el cual es flexible y puede manejar cualquier tipo de información. El host usa las transferencias de tipo control y de interrupción para enviar y recibir información y pedir reportes.

Cada transacción está limitada en la cantidad de información que pueda transferir, de acuerdo al tipo de dispositivo el máximo es de:

- 8 bytes para los de velocidad baja
- 64 bytes para los de velocidad media
- 1024 bytes para los de velocidad alta

Un dispositivo puede enviar información a la computadora en cualquier momento, por ejemplo un clic de ratón, por lo que el host inspecciona al dispositivo periódicamente.

La velocidad máxima de transferencia está limitada, especialmente en los dispositivos de media y baja velocidad. El host garantiza un máximo de:

- 800 bytes/segundo para los de velocidad baja
- 64000 bytes/segundo para los de velocidad media
- 24576 bytes/segundo para los velocidad alta

No se garantiza un taza de transferencia fija sobre todo en los dispositivos de baja velocidad.

3.2.8.1 IMPLEMENTACIÓN DE CONTROLADOR USB

Para implementar cualquier controlador de hardware para el sistema operativo Windows 95/98/2000/XP, Microsoft ha desarrollado la herramienta para el desarrollo de manejadores de nombre WinDDK (Windows Driver Developer Kit) el cual se puede

solicitar desde su página web http://www.microsoft.com/winddk. Este software trabaja en conjunto con la herramienta de programación Visual Studio 6.0, en donde se puede trabajar con cualquiera de sus lenguajes de programación que nos ofrece, en nuestro caso Visual C++ pues la mayor cantidad de ejemplos e información que encontramos en la red referentes al desarrollo de un manejador USB estaban hechos en este lenguaje.

Para seguir con la implementación se debe tener en cuenta con qué versión de Windows se va a trabajar, porque dependerá de esto las librerías que se empleen del WinDDK.

La ventaja de usar el WinDDK es que nos da más opciones en cuanto a la diversidad de librerías que nos brinda y a su vez mayor facilidad en el desarrollo, en comparación a tener que partir casi desde cero en el caso de hacerlo en Linux. Con esta ventaja se puede desarrollar un controlador completamente personalizado con un ID propio, tocando desde luego el ID del firmware del dispositivo. Este caso puede ser ventajoso por si queremos implementar un dispositivo que no tenga ningún conflicto con algún dispositivo HID, para lo cual basta con seguir el esqueleto del controlador, cargar las librerías necesarias, compilar el programa y probarlo, para esto último se cuentan con programas de evaluación del controlador como son el USB Driver Tester que viene incluido en el paquete del compilador CCS utilizado en nuestro firmware, o el USB Command Verifier del foro de Implementación de USB.

CAPÍTULO IV - DESARROLLO Y RESULTADOS DEL PROYECTO

Tal y como se ha explicado durante el desarrollo de esta tesis, el presente trabajo ha sido realizado para satisfacer en primera instancia cuatro factores actuales que se presentan en el EEG Digital desarrollado en CIDESI:

1. Reducción de procesos que realiza la PC
2. Determinismo en la aplicación
3. Comunicación con cualquier computadora personal
4. Disminución de costos

En cada capítulo presentado hasta este momento se han proporcionado las bases o teoría revisada, analizada y utilizada en el presente trabajo para poder lograr los objetivos planteados al inicio del proyecto.

A continuación se expondrá cada parte del desarrollo de este trabajo y los resultados obtenidos. Se parte de la premisa que existe ya un equipo de electroencefalografía digital desarrollado previamente en CIDESI, el cual cuenta de un cabezal o módulo de amplificación de las señales bioeléctricas del cerebro y una computadora personal PC, la cual contiene una tarjeta de adquisición de datos comercial y un software para la adquisición, procesamiento, despliegue y almacenamiento de las señales bioeléctricas.

Sobre esta base, el presente trabajo se enfoca en la modificación de la adquisición de las señales y parte del software actual. Esto es, se sustituye la tarjeta de adquisición de datos comercial por un desarrollo propio, se elimina de la PC el procesamiento de las señales y se añade la comunicación hacia el amplificador de señales por medio del puerto USB.

4.1 ADQUISICIÓN DE LAS SEÑALES DE EEG

Primeramente se revisa la adquisición de los datos, la secretaría de salud mediante el cuadro básico para equipos de electroencefalografía digitales, requiere que la frecuencia de muestreo sea de 1000 datos por segundo y la cantidad de datos que se requieren monitorear son 21 electrodos, cada electrodo es un canal a adquirir.

Además se conoce que la señal de interés de electroencefalografía va desde 25 a 200 µV de amplitud y de 0.5 a 100 Hz de frecuencia.

Con la información anterior se realiza una búsqueda de convertidores análogo-digital los cuales se pudieran utilizar en la aplicación, esta búsqueda se realiza considerando también el tamaño final del prototipo.

Primeramente se busca convertidores en Texas Instruments considerando el número de canales disponibles por componente, esta empresa maneja convertidores analógicos a digital de 1, 2, 3, 4, 6, 7, 8, 11, 12 y 16 canales. Si seleccionamos uno de 8 canales podríamos tener resolución hasta de 24 bits pero se requiere tres de estos convertidores. Si se selecciona el de 11, tenemos hasta 12 bits de resolución y solo ocuparíamos dos de ellos. Con las opciones restantes también necesitaríamos dos convertidores, por lo que se reduce las opciones a convertidores de 8 y 11 canales.

Adicionalmente se revisa la resolución del convertidor, como ya se mencionó para convertidores de 8 canales podremos tener una resolución máxima de 24 bits y para uno de 11 canales sólo tendremos como máximo 12 bits. Ahora entonces se revisa los bits necesarios para tener una resolución aceptable en nuestro sistema.

Sabemos que la señal de interés de electroencefalografía tiene una amplitud máxima de 200 µV o ±100 µV para un acondicionamiento bipolar. Y debido a que el DSP es alimentado a 3.3 Volts manejamos para el ADC una referencia de 1.5 Volts, con esto se obtiene que para -100 µV tendremos después de las etapas de amplificación una señal de 0 V y para una de 100 µV se tendrá 3 V.

Con los datos anteriores calculamos la resolución para 12 bits. Aplicando la fórmula 4.1.

$$Resolución\ de\ la\ señal = \frac{Rango\ señal\ de\ entrada}{2^{nbits}-1} \tag{4.1}$$

Sustituyendo valores tenemos

$$\frac{3-(0)}{2^{12}-1} = 733\mu V \tag{4.2}$$

Lo anterior nos indica que por cada variación de bit en el convertidor tenemos 733 µV de variación a la entrada, este valor es el mínimo voltaje que nuestro sistema puede detectar con esta resolución. Este valor es después de una amplificación de la señal de interés, por lo que falta calcular, cuál sería la mínima variación de la señal original que

el sistema podría detectar, mediante calculo tenemos que sería aproximadamente 0.049 µV, lo suficientemente bueno para nuestra aplicación.

Basándonos en estos cálculos, se considera para el diseño utilizar el ADC TLV2556, de la empresa Texas Instruments. El cual tiene las siguientes características [48]:

- 12 Bits de resolución.
- Hasta 200 kS/seg.
- 11 canales de entrada analógica.
- Máximo error de linealidad ±1 LSB.
- Operación unipolar o bipolar
- Longitud de datos de salida programables.
- Interfaz de comunicación serial SPI hasta 15 MHz.

La figura 4.1, muestra el diagrama funcional del ADC TLV2556 utilizado. Los pines AIN0 hasta AIN10 son las entradas analógicas, VCC es conectado a 3.3 V, se maneja en modo unipolar, por lo que REF+ es conectado a una referencia de 1.5 V y REF- es conectado a GND (0 V), $\overline{\text{INT}}$/EOC genera una interrupción al DSP indicando que se ha completado una conversión.

La comunicación SPI es implementada con los pines:

- DATA IN, por el cual el DSP configura al ADC para trabajar.
- $\overline{CS}$ por medio de este pin el DSP selecciona cual de los dos ADC le solicita datos.
- I/O CLOCK por medio de este pin el DSP le proporciona al ADC la frecuencia de reloj para su operación.
- DATA OUT el ADC le envía al DSP la trama de datos con los diez valores de conversión al DSP.

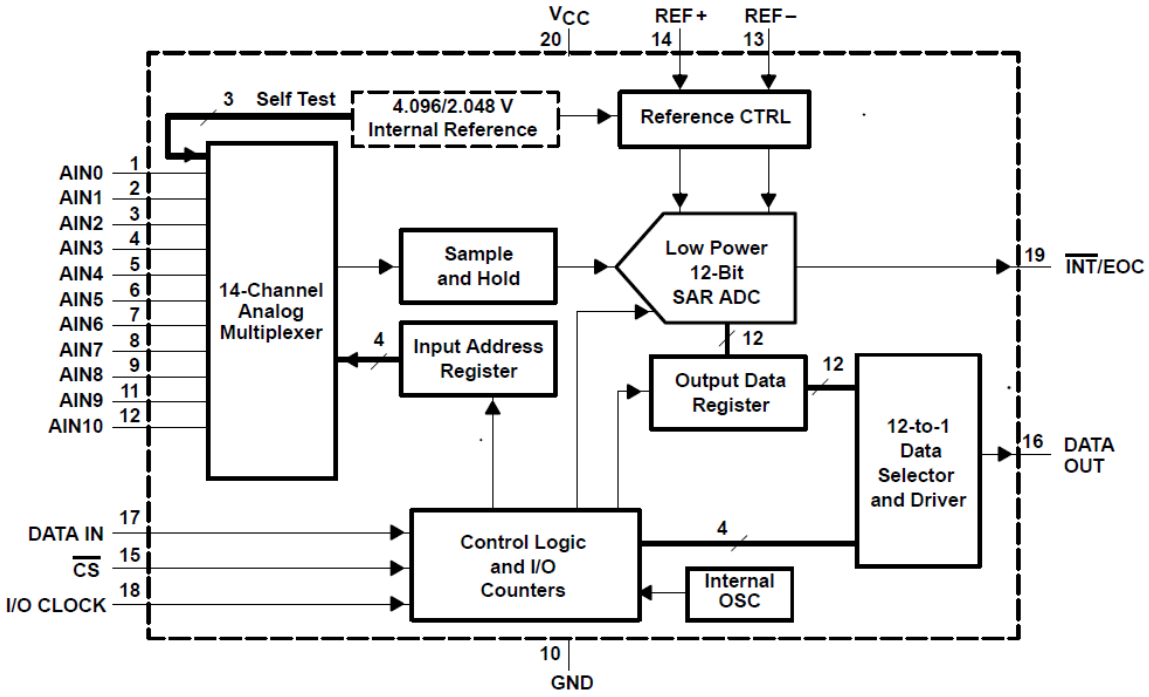


Figura 4.1 Diagrama de bloque funcional del ADC.

4.2 DISEÑO DE FILTROS DIGITALES

Debido al tipo de señal que se maneja en la aplicación, la cual es un biopotencial eléctrico del cerebro, se tiene dos aspectos a considerar para el desarrollo del filtrado de la señal.

En primer lugar, debido al movimiento de los cables que llevan las señales desde el paciente hasta el equipo EEG, se genera un offset en la señal adquirida del cabezal, el cual tiene una frecuencia de oscilación por debajo de los 0.5 Hz.

En segundo lugar, se sabe que las señales de interés para esta aplicación están por debajo de los 100 Hz y que por cuestiones de alimentación eléctrica en México contamos con un suministro de energía de 60 Hz, por esta razón en nuestras señales adquiridas se tiene siempre presente esta señal de 60 Hz, que debido a las magnitudes de la señal de interés muchas veces el ruido de 60 Hz es mayor a la señal del EEG.

En base a estas dos consideraciones, se requiere la implementación de 3 filtros básicamente. El primero sería un filtro pasa-altas con una frecuencia de corte de 0.5 Hz, el segundo sería un filtro pasa bajas con una frecuencia de corte de 100Hz y por último

un filtro rechaza-banda de 60 Hz. Con esta información se empieza a trabajar en el diseño de cada uno de estos filtros para después implementarlos en el DSP.

Es importante mencionar que gracias a la experiencia que se tiene en el lenguaje de programación LabVIEW, el diseño y pruebas de implementación de los filtros digitales se realizaron bajo esta plataforma, una vez que los filtros han sido validados se realiza su implementación en el DSP.

Para tomar una decisión correcta sobre el filtrado digital, antes de realizar su implementación en el DSP, se procede a trabajar en una PC con señales obtenidas de una base de datos mundial de señales fisiológicas, donde se incluyen señales de EEG. Esta base de datos es Physionet [53] y actualmente contiene más de 80,000 grabaciones de señales fisiológicas organizadas en 75 bases de datos.

Se descarga una colección de datos de señales de electroencefalografía, cabe señalar que las señales de esta colección de datos son filtradas y adquiridas de equipos de electroencefalografía. En la figura 4.2 se muestran los datos con los cuales se trabaja en esta parte del desarrollo, cada grupo de datos corresponden a 10 segundos de grabación.

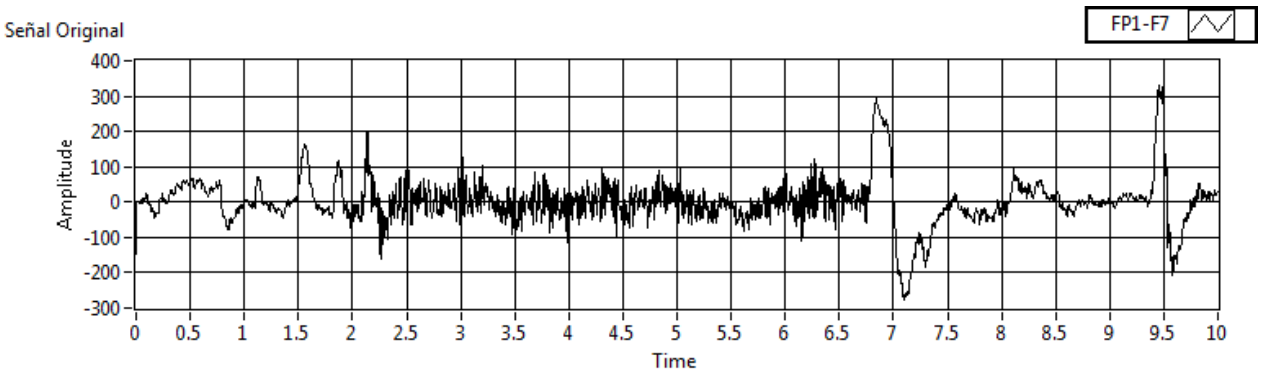


Figura 4.2 Señal Fp1-F7 de Physionet.

Como ya se ha mencionado anteriormente, las señales del EEG pueden tener interferencias de baja frecuencia, ruido de 60 Hz y posiblemente algo de interferencia de alta frecuencia. Por lo tanto se modifica la señal original obtenida de la base de datos, como se observa en la figura 4.2, sumándole señales, simulando el ruido de baja

y alta frecuencia, así como también el de 60 Hz. En la figura 4.3 se muestra la señal Fp1-F7 ya con el ruido añadido a la señal original.

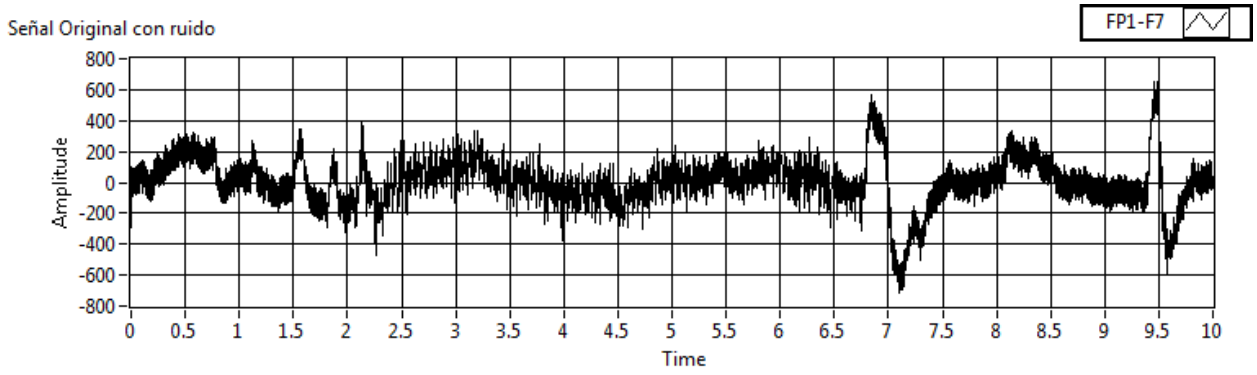


Figura 4.3 Señal Fp1-F7 de Physionet con ruido.

De acuerdo a la señal mostrada anteriormente, requerimos un filtro pasa-banda que va de muy baja frecuencia, dígase 0.5 o 1 Hz, hasta una frecuencia de 100 Hz aproximadamente y además un filtro rechaza-banda de 60 Hz. El filtro pasa-banda nos ayudará a eliminar la señal de muy baja frecuencia inducida por el movimiento de los cables de EEG y el offset presente en la señal. También nos ayudará a eliminar las altas frecuencias, las cuales no son parte de nuestra señal de EEG. Y como la señal de línea de 60 Hz cae dentro de nuestro rango de medición tendremos que eliminar esta frecuencia también, utilizando el rechaza-banda.

En LabVIEW existe un VI (función) llamado, Configure Classical Filter Design, ver figura 4.4. Este VI nos permite generar el arreglo de coeficientes de acuerdo al diseño de filtro requerido.

Para realizar el filtro pasa-banda tenemos dos opciones, la primera es realizar un filtro pasa-bajas en cascada con un filtro pasa-altas o viceversa y la segunda es diseñar directamente el filtro pasa-bandas. En base a lo anterior, se realizan diferentes pruebas en LabVIEW para verificar cuál de estas configuraciones se tiene mejor resultado.

Para decidir cómo implementar el filtro pasa-banda, primero se verifica cuál de las dos opciones anteriormente mencionadas cuenta con un orden de filtro menor y por supuesto que tenga un comportamiento adecuado con una señal real de EEG. El orden de un filtro nos indica cuantos coeficientes tiene. Y entre más coeficientes tenga un

filtro, su implementación requiere mayor procesamiento, por esta razón se buscaría la implementación con el menor número de coeficientes posible.

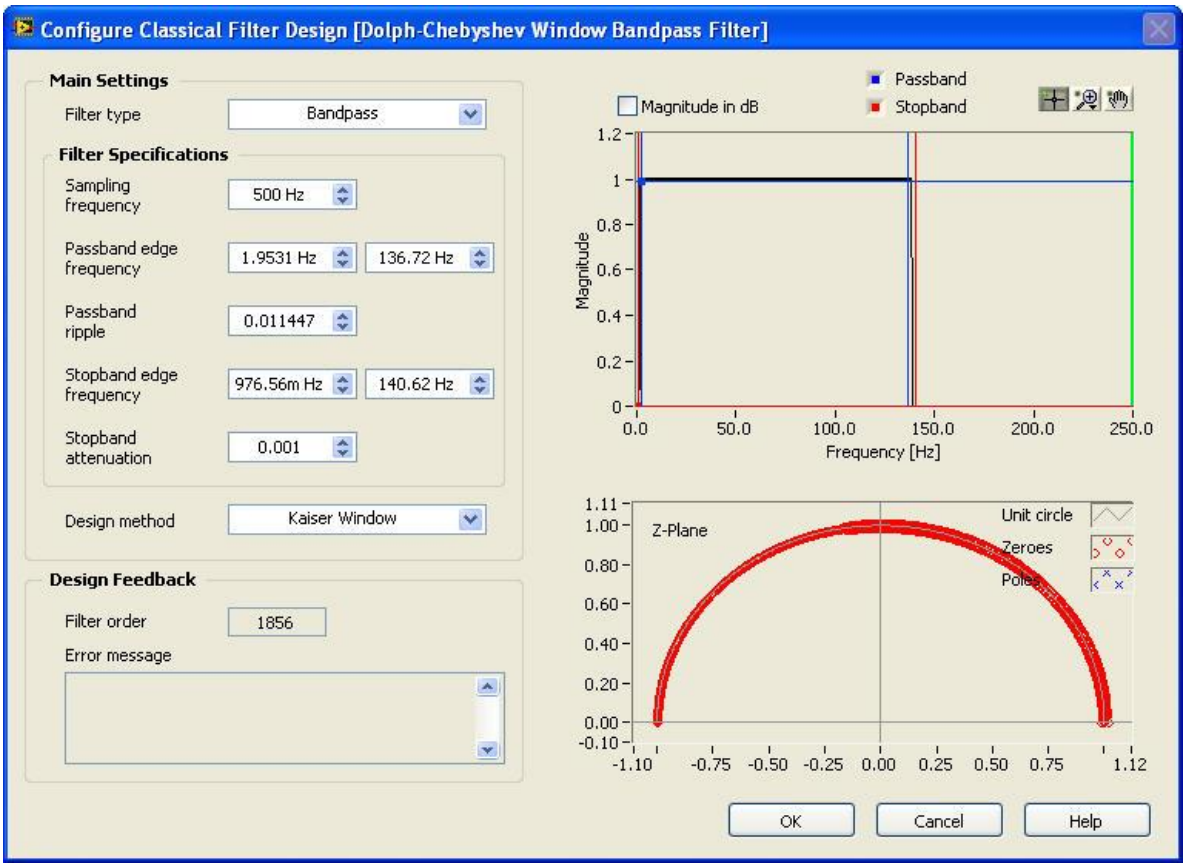


Figura 4.4 Diseño de filtro pasa-banda.

Utilizando el VI “Configure Classical Filter Design”, seleccionamos un filtro pasa-banda directamente, y obtenemos que el filtro tendría 1856 coeficientes, acorde a la configuración realizada. Por el contrario al seleccionar un filtro pasa-bajas y uno pasa-altas tendríamos un orden de 474 y otro 1856 para un total de 2330, con la misma configuración que el filtro pasa-banda.

De la misma manera se realiza el diseño para el filtro rechaza-banda, utilizando la herramienta de LabVIEW, en la figura 4.5 se muestra la configuración realizada, obteniendo un filtro de orden 800.

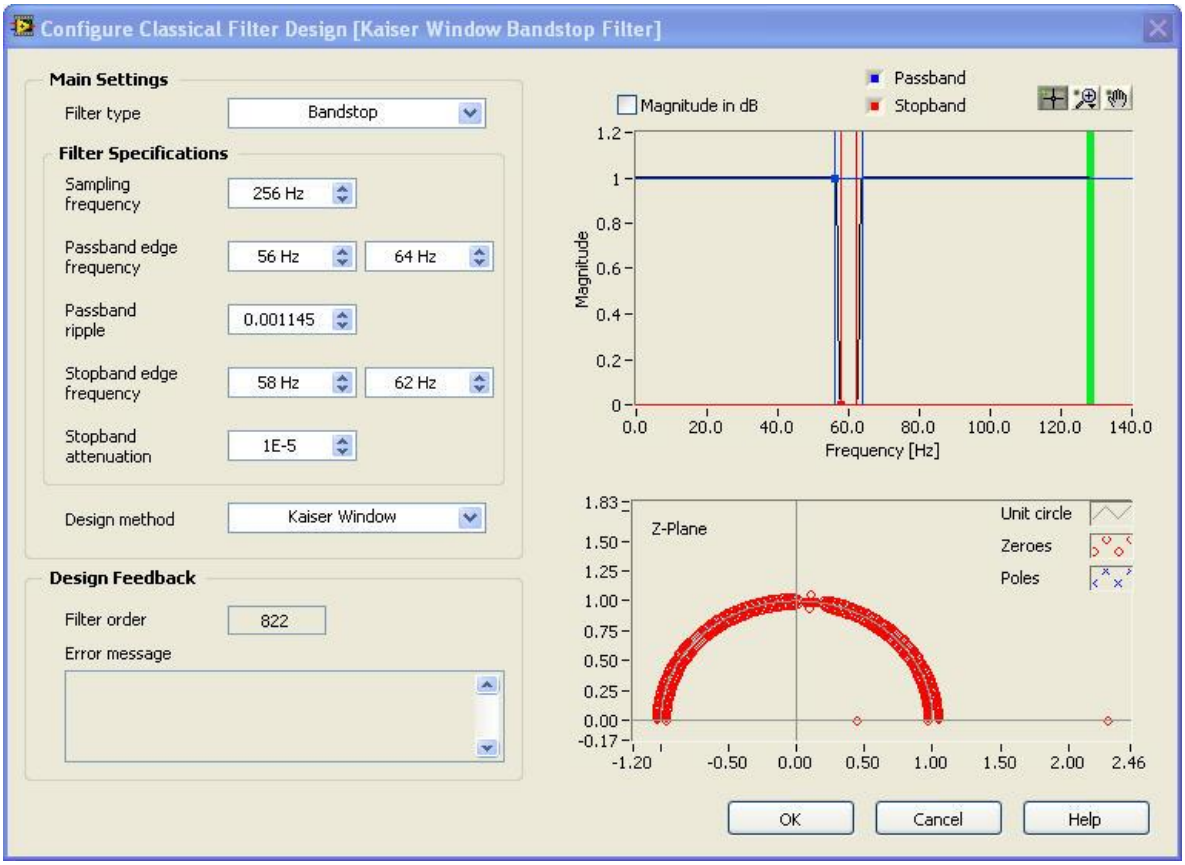


Figura 4.5 Diseño de filtro rechaza-banda.

Como lo comentamos previamente, ahora requerimos validar la funcionalidad de ambas implementaciones para un filtro pasa-banda. En las figuras 4.6 y 4. 7 se muestran las imágenes de las señales filtradas por ambas implementaciones (pasa-banda directo o pasa-altas + pasa-bajas) en conjunto con el filtro rechaza-banda.

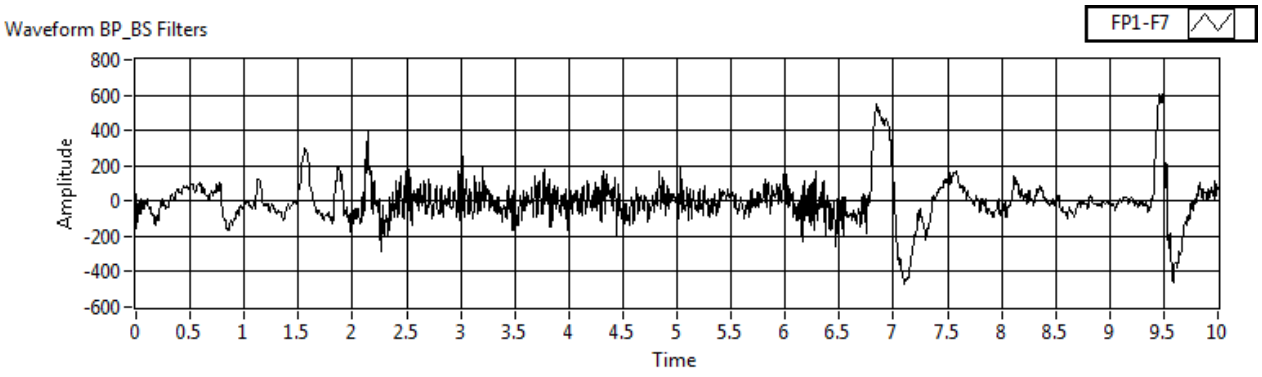


Figura 4.6 Señal Filtrada por filtro pasa-banda y rechaza-banda.

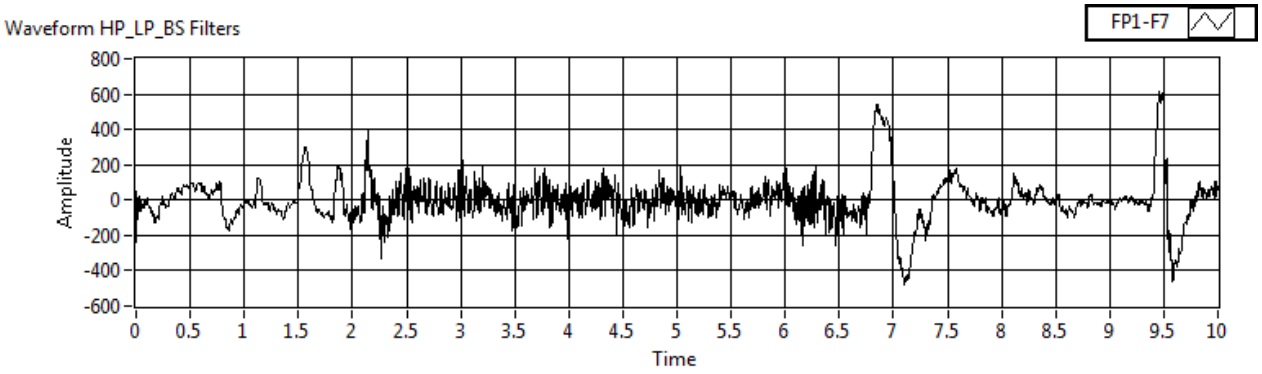


Figura 4.7 Señal Filtrada por filtro pasa-altas en casacada con pasa-bajas, así como rechaza-banda.

Adicional a estas dos implementaciones, también se validó una tercera, en donde se intercambian los filtros pasa-altas y pasa-bajas, pero al tener el mismo comportamiento que las dos implementaciones anteriores no se coloca la imagen de la señal resultante con esta última implementación.

En la figura 4.8 se muestra parte del código implementado en LabVIEW para la realización de estas pruebas.

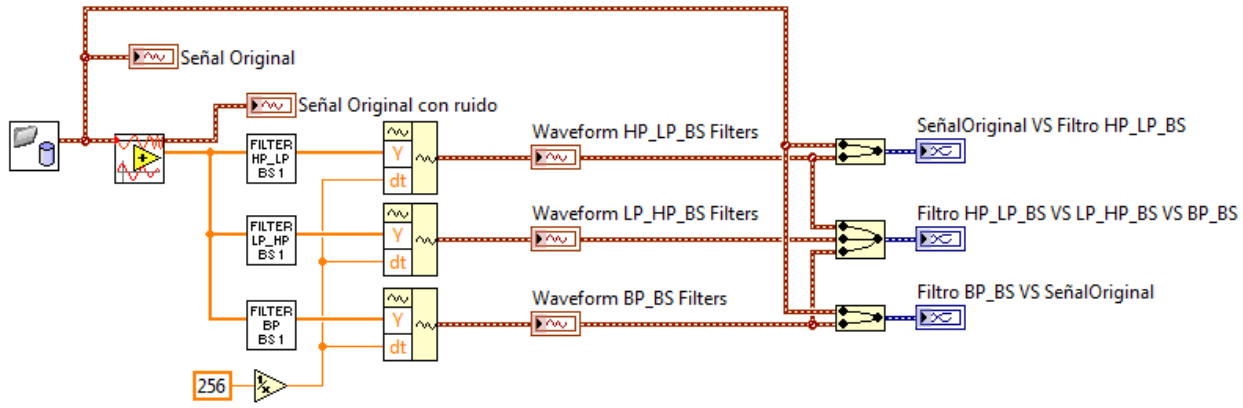


Figura 4.8 Código LabVIEW – Análisis de Filtros.

De los resultados obtenidos se puede observar que prácticamente no existe variación entre las diferentes implementaciones posibles, por lo que se considera implementar el filtro pasa-banda directamente en el DSP, en lugar de hacer el filtro en cascada. En la figura 4.9 se muestra las dos señales sobre puestas (pasa-banda directo y pasa-altas en cascada con pasa-bajas).

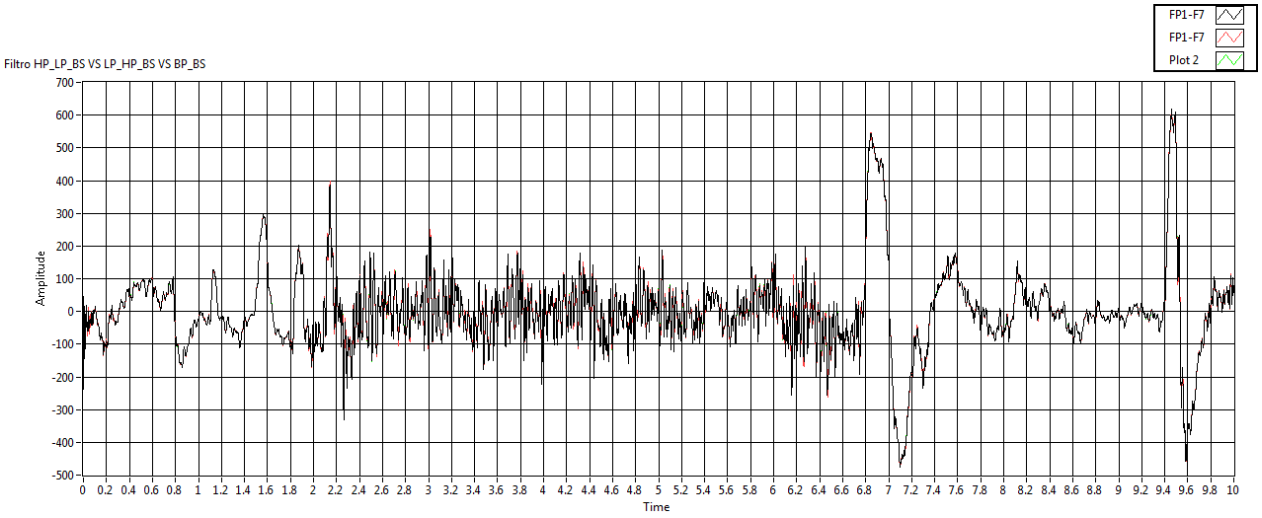


Figura 4.9 Señales superpuestas del filtro pasa-banda.

Los coeficientes generados para cada filtro son almacenados en un archivo de texto, el cual se utilizará ya en nuestro diseño final para la implementación del filtro digital en el DSP.

4.3 PROCESADOR DIGITAL DE SEÑALES TMS320C5509A

Ahora empezaremos a trabajar con el DSP, el procesador de señales utilizado en este trabajo en conjunto con el ADC es el TMS320VC5509 de la empresa Texas Instruments, inicialmente se utiliza una tarjeta de evaluación de este procesador fabricada por la empresa Spectrum Digital TMS320VC5509EVM, a partir de esta tarjeta se empezó a trabajar hasta tener un prototipo propio de CIDESI.

La tarjeta de evaluación (EVM) mencionada nos permite examinar todas las características del procesador de señales digitales C5509, para poder determinar si éstas empatan con los requerimientos deseados para esta aplicación, como son: bajo consumo de energía, comunicación USB y determinismo. Además este módulo es una excelente plataforma para desarrollar y ejecutar firmware para los procesadores de señales digitales TMS320VC5509 [49].

En la figura 4.10, se muestra el diagrama a bloques de la tarjeta de evaluación utilizada y a continuación también se mencionan algunas de las características generales de dicha tarjeta de evaluación.

- Procesador VC5509A operando a 120 MHz.
- Interfaz de Bus Serial Universal (USB) 1.1
- Display de cristal líquido (LCD) de 128x64 pixeles y teclado con nueve botones y potenciómetros.
- Cuatro conectores de expansión (datos, entradas/salidas, control e interfaz de host port).
- Interfaz para memorias tipo Memory Stick y Media Card.
- Memoria RAM 4 Meg x 16 DRAM
- Memoria Flash 1 Meg x 16.
- Conector JTAG (1149.1) para emulación.
- Operación a 5 VDC.

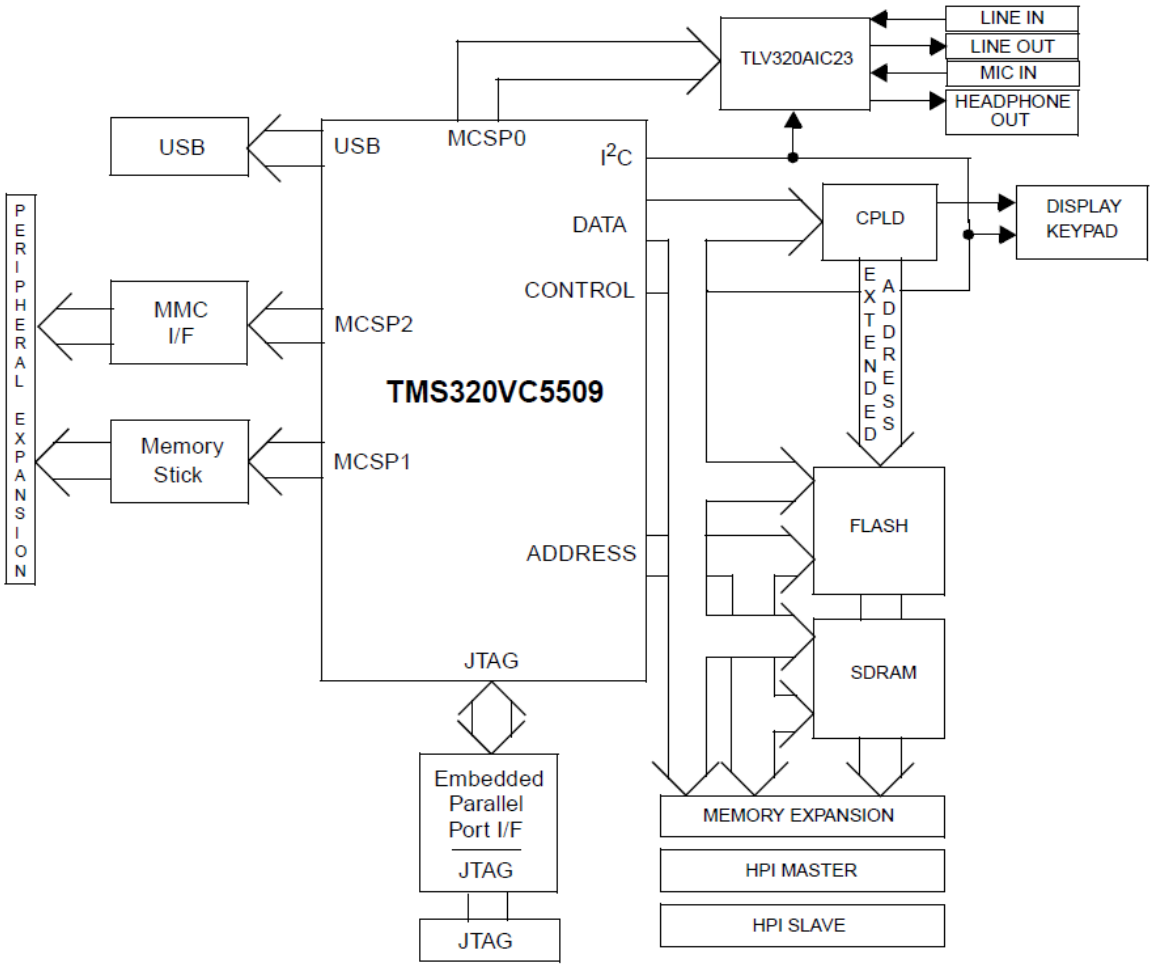


Figura 4.10 Diagrama a bloques TMS320VC5509 EVM.

Ahora realizaremos el análisis para determinar la cantidad de datos que se requieren transmitir del DSP a la PC acuerde a las características técnicas de nuestro sistema.

Como se mencionó previamente el equipo debe ser capaz de trabajar a frecuencias de muestro que van de 200 Hz a los 1000 Hz, lo anterior por norma del cuadro básico para equipos EEG registrados en el país. Por lo tanto si tenemos una fs=200, nuestro ADC tiene 12 bits de resolución y el amplificador del EEG tiene 21 canales. Entonces, tendremos:

$$fs * \#bits_{ADC} * \#Canales = 200 * 12 * 21 = 50{,}400\ bits/seg \qquad (4.3)$$

Si se cambia únicamente la frecuencia de muestreo a 1000 Hz, la cual corresponde a la frecuencia de muestreo más alta contemplada para la aplicación, tendremos:

$$fs * \#bits_{ADC} * \#Canales = 1000 * 12 * 21 = 252{,}000\ bits/seg \qquad (4.4)$$

De acuerdo a los objetivos planteados, para una adquisición de 1000 datos/seg tenemos un requerimiento de 252 kbits/seg. El procesador digital de señales C5509A cuenta con un puerto de comunicación USB con el cual es posible desarrollar dispositivos USB del tipo "*full speed"*, el cual es completamente compatible con la versión 1.1 de la especificación del bus serial universal (USB) con una taza de transferencia de datos de hasta 12Mb/s. Así se logra la transferencia máxima de datos utilizando el puerto USB.

Para el desarrollo del firmware (programación, compilación y depuración) en el DSP utilizamos el software Code Composer Studio v3.1, éste es un ambiente de desarrollo integrado (IDE) exclusivo para dispositivos de la empresa Texas Instruments.

Como ya se ha mencionado anteriormente, la aplicación requiere adquirir, filtrar y enviar datos hacia una computadora personal, donde el tiempo entre cada tarea ejecutada se torna crítico en la aplicación, por lo tanto se decidió utilizar una herramienta muy poderosa de los DSP's de Texas Instruments, la cual es llamada DSP/BIOS. Esta herramienta es un kernel de tiempo real escalable, la cual nos provee de métodos para sincronizar y calendarizar tareas críticas en forma paralela [50].

Por lo que mediante este kernel o sistema operativo se controla los tiempos de adquisición y procesamiento de datos, además nos permite manejar adecuadamente la memoria del dispositivo, tanto en la definición del tamaño como en las transferencias de datos por DMA (Acceso Directo a Memoria).

En la figura 4.11, se muestra el diagrama de estados del firmware desarrollado en el DSP, donde podemos apreciar cómo se ejecutan tareas en forma paralela.

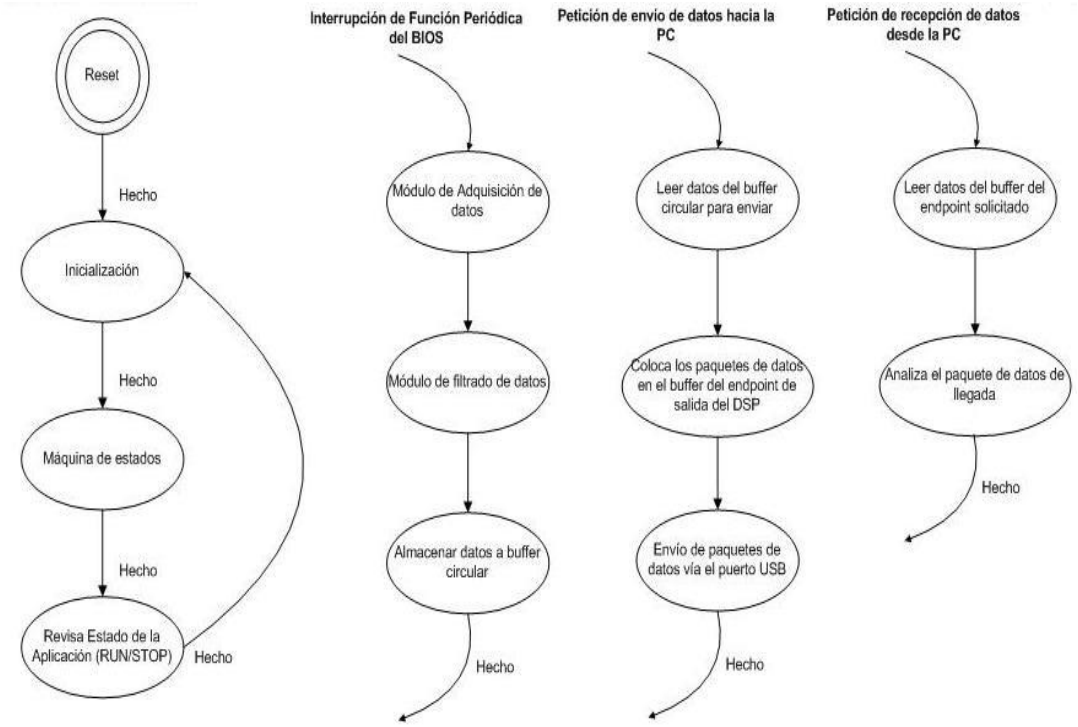


Figura 4.11 Diagrama de estados del firmware desarrollado

En conjunto con este kernel, durante este trabajo se utiliza otra herramienta importante de Texas Instruments, llamada DSP/BIOS Driver, esta herramienta nos permite desarrollar drivers específicos para periféricos presentes en el DSP y sus tarjetas de evaluación asociadas o prototipos finales personalizados, además, si no se requiere desarrollar drivers nuevos, también nos permite integrar a una aplicación drivers previamente desarrollados por otros [52]. En nuestro caso utilizamos esta herramienta como punto de partida para manejar la tarjeta de evaluación del DSP y así lograr un entendimiento del manejo de sus recursos y también para el desarrollo de un driver propio para la comunicación con los ADC’s utilizados en este trabajo.

En la figura 4.12, podemos visualizar el entorno de desarrollo para el firmware del DSP. El código mostrado corresponde a la implementación de la adquisición de datos usando comunicación SPI con el ADC externo.

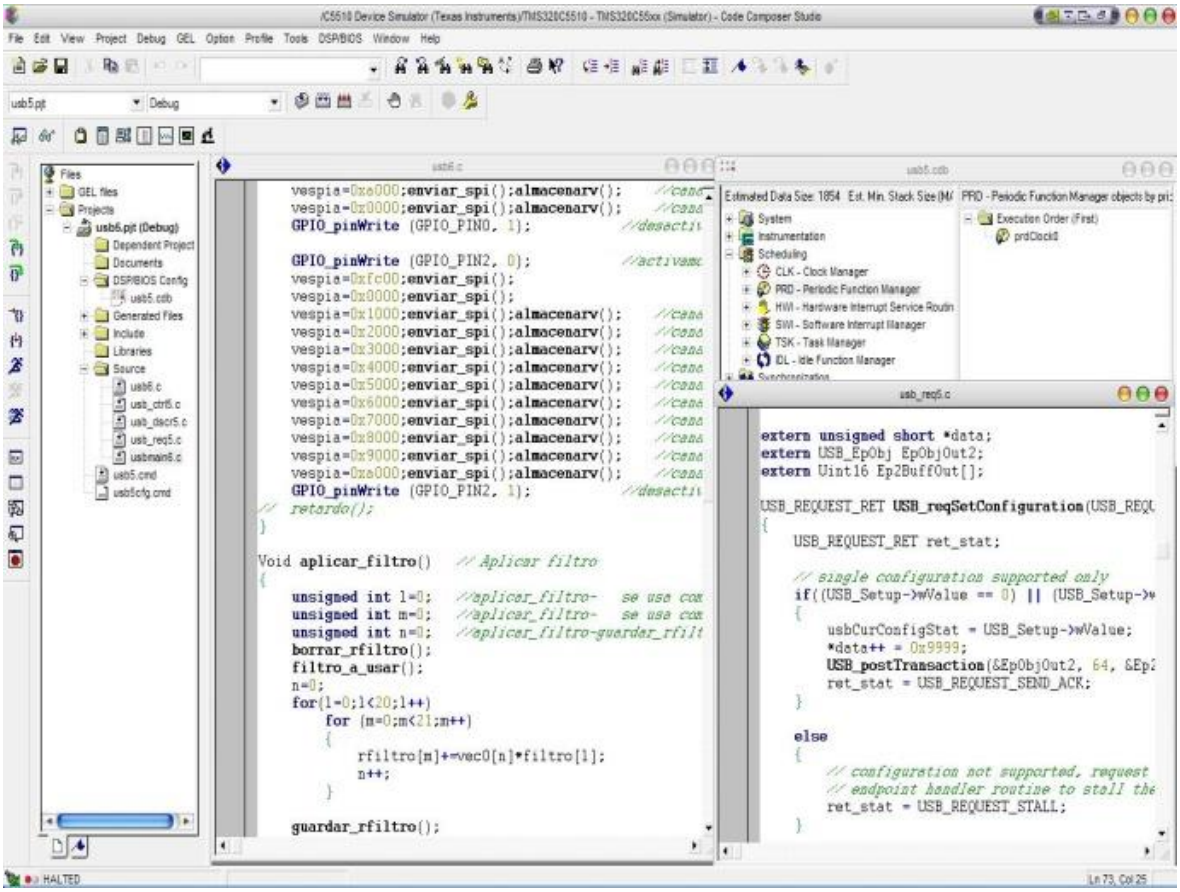


Figura 4.12 Entorno de desarrollo para el DSP.

Durante la etapa de diseño y depuración del código del firmware del DSP, se trabajó con señales sinusoidales generadas con un equipo comercial y a su vez a estas señales se le agregaron diferentes niveles de ruido, variando su amplitud y frecuencia.

Primeramente se trabajó con la adquisición de datos, generando un driver para la comunicación con el ADC seleccionado y el DSP 5509A. Este módulo de adquisición se trabajó hasta dejarlo funcional para diferentes frecuencias de muestreo que van desde los 200Hz hasta los 1000 Hz, considerando que esta frecuencia de muestreo será seleccionada por el usuario mediante el software en la PC, en valores ya preestablecidos. Las opciones finales serían 200Hz, 500Hz y 1000Hz.

Una vez validada la adquisición de datos, se continuó con el módulo para el filtrado de la señal. En este punto se implementaron en el DSP dos filtros digitales tipo FIR, mediante la convolución de los coeficientes obtenidos por LabVIEW y los datos adquiridos para cada uno de ellos.

En este tiempo, básicamente se realizaron diferentes pruebas para validar los diseños de ambos filtros, primeramente se aplica el filtro pasa-banda y a continuación el rechaza-banda. Se trabajó con diferentes órdenes de filtros, ya que originalmente en el diseño de éstos se obtuvieron órdenes de filtro muy grandes y esto implicaba la realización de más operaciones y por lógica un incremento en el tiempo de procesamiento. En forma general a mayor cantidad de coeficientes mayor es el tiempo que se le dedica al procesamiento de los datos.

Por lo tanto, se trabaja estrechamente con el diseño en la PC e implementación del filtro en el DSP hasta llegar a un balance entre cantidad de coeficientes y tiempo de procesamiento, para así llegar a un punto óptimo de acuerdo a la señal de interés.

Finalmente se deja un tamaño para los coeficientes del filtro pasa-banda de 150 y para el rechaza-banda se deja de 55.

La validación de las frecuencias de corte de ambos filtros se realizó apoyándonos de un generador de frecuencia, ver figura 4.13, el cual proveía de las señales eléctricas al amplificador del EEG, y variando la frecuencia de la señal generada se visualiza en la aplicación la atenuación de la señal para frecuencias anteriores y posteriores de las de corte.

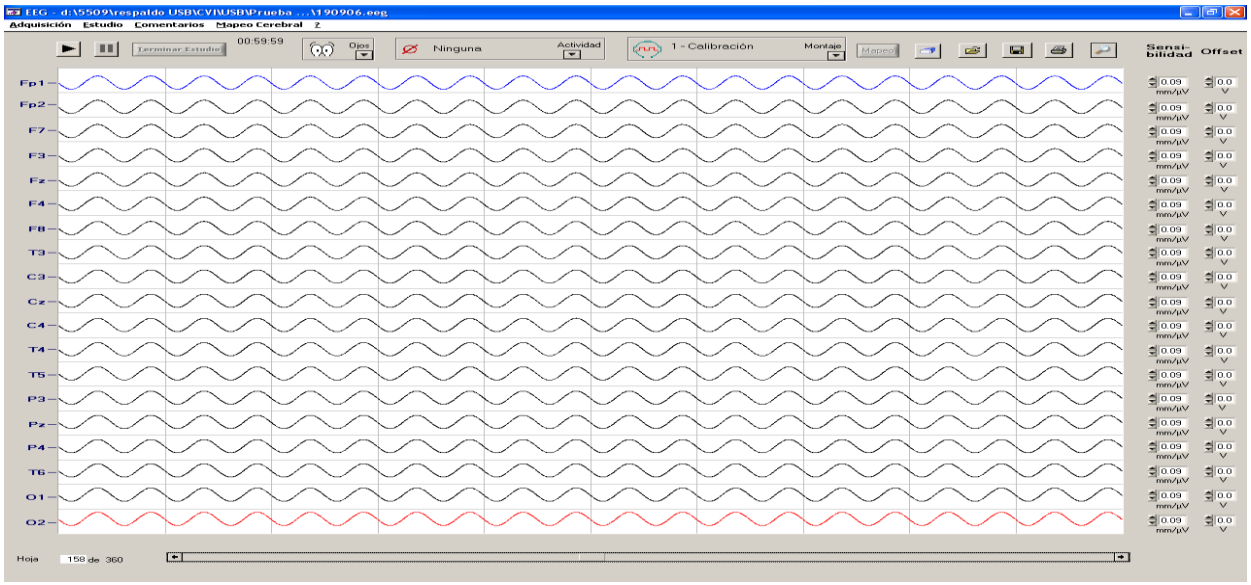


Figura 4.13 Validación de filtrado digital implementado en DSP.

4.4 COMUNICACIÓN USB

Una vez que las etapas de adquisición y procesamiento han sido validadas en el DSP, se procede a trabajar con la comunicación USB. Es importante mencionar que para trabajar con el puerto USB se requiere desarrollar en conjunto tanto el código en el DSP como en la PC.

Además del kernel y de la herramienta para generación y utilización de drivers, mencionadas previamente, también se utilizó una librería de gran ayuda llamada Chip Support Library. Esta herramienta agrupa un set de funciones, macros y símbolos usados para la configuración y control de los periféricos del procesador [51]. Con esta librería empezamos a trabajar el envío de datos desde el DSP a la PC mediante el puerto USB, ya que nos abstrae en gran medida del manejo del hardware. Pero debido a problemas que se presentaron durante el desarrollo en base a la comunicación USB, se tuvo que realizar cambios a funcionalidades de las API, y para lograr esto se tuvo que entender a bajo nivel como realizar la implementación de la comunicación USB.

El DSP C5509A nos permite crear un dispositivo esclavo USB full speed, ya que solo es compatible con la versión 1.1 y 2.0, este dispositivo no cumple con las especificaciones para USB 3.0.

En un sistema USB, hasta la versión 2.0, se tiene una comunicación de un dispositivo maestro a un esclavo, normalmente el host es el dispositivo maestro y éste es quien inicia todas las transferencias de datos entre el mismo y los dispositivos USB agregados al sistema. Por lo tanto es importante entender como es la dirección de una transferencia de datos, la cual es descrita de manera relativa al host:

- Transferencia de Salida (OUT Transfer): Se refiere a una transferencia de datos desde el host hacia el dispositivo, Host → Dispositivo.
- Transferencia de Entrada (IN Transfer): Se refiere a una transferencia de datos desde un dispositivo hacia el host, Dispositivo → Host.

En nuestro caso el Host se refiere a la PC en la cual se conectará el EEG mediante el puerto USB, y el dispositivo es la tarjeta desarrollada en este trabajo para comunicar el EEG con la PC.

Como se explicó en el capítulo 3 de este trabajo, cada transferencia de una comunicación USB, ya sea de entrada o salida, puede ser de alguno de los siguientes tipos: Transferencia de Control (Control Transfer), Transferencia Bulk (Bulk Transfer), Transferencia de Interrupción (Interrupt Transfer) o Transferencia Isócrona (Isochronous Transfer).

El dispositivo utilizado en este trabajo tiene un endpoint de entrada y un endpoint de salida para cada uno de los tipos de transferencia que se pueden manejar en una comunicación USB. Todas las transferencias de datos entre el host USB y el dispositivo USB, los datos pasan siempre a través de un endpoint en el dispositivo.

De acuerdo a lo mencionado anteriormente tenemos:

- OUT Endpoint: Se refiere a un endpoint que mantiene los datos recibidos desde el host USB. Para usar los datos que la PC envía, el DSP debe leer los datos desde un endpoint de salida.
- IN Endpoint: Se refiere a un endpoint que mantiene los datos que se enviarán al host USB. Para enviar los datos adquiridos hacia el host, el DSP debe escribir los datos a un endpoint de entrada.

El dispositivo USB utilizado en este desarrollo tiene 16 endpoints, dos de estos endpoints son exclusivos para las transferencias de control (IN0 y OUT0). Y catorce endpoints de propósito general (IN1-IN7 y OUT1-OUT7) los cuales se pueden configurar por código para cualquiera de las otras tres transferencias (Bulk, Interrrupt e Isochronous).

En la figura 4.14 se muestra de manera conceptual cada uno de los bloques del módulo USB presentes en el DSP C5509A, los bloques sombreados no son parte del módulo USB pero si del DSP. El bloque denominado SIE es el manejador del protocolo de USB. Así pues en esta figura se muestra la interacción que existe entre los diferentes bloques del módulo de USB con el CPU y la memoria del DSP para llevar a cabo transferencias tanto de entrada como de salida de datos vía el puerto USB.

Es ampliamente recomendable el uso del controlador DMA, para transferir datos entre la memoria del DSP y el módulo de USB, para así reducir el consumo de tareas al procesador, debido a que las transferencias por DMA son mucho más efectivas, que transferir directamente los datos de la memoria del DSP sin el uso de DMA. Por lo tanto en la implementación en el DSP, se realiza el manejo de transferencias a través del DMA.

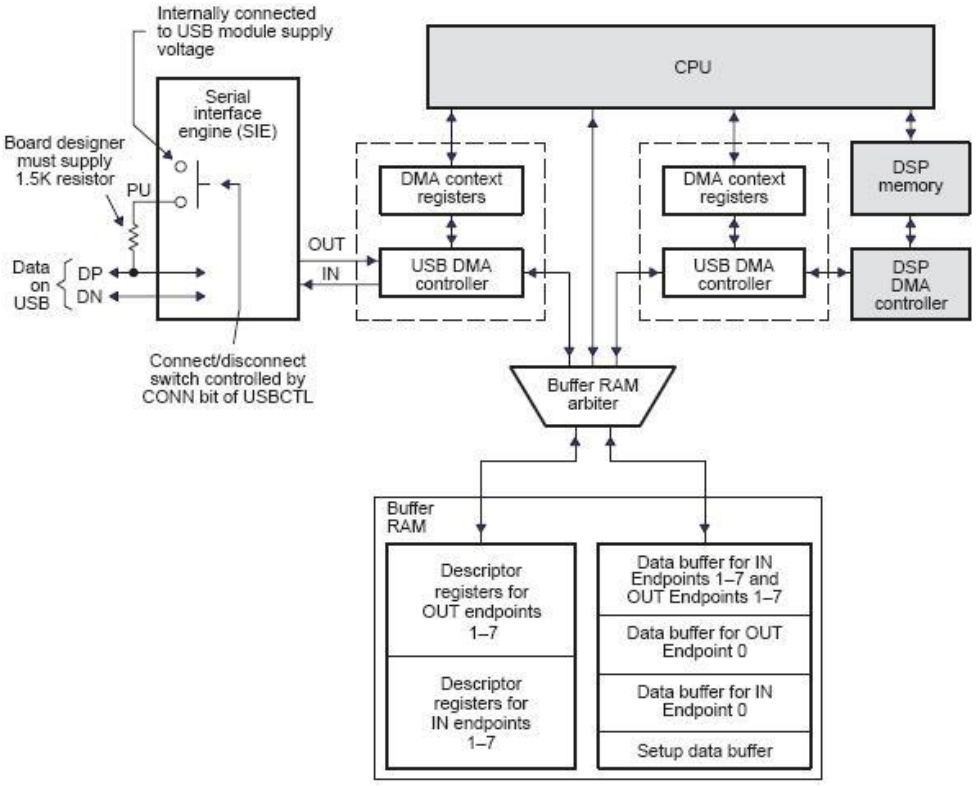


Figura 4.14 Diagrama a bloques conceptual del módulo USB del DSP C5509A.

Acorde a la teoría mostrada en el capítulo 3 y a las características técnicas de nuestra aplicación se requiere que la comunicación de los datos adquiridos hacia la PC se realice utilizando la transferencia isócrona. Esta decisión se basa en que este tipo de transferencia garantiza la menor latencia posible en la comunicación entre el host y el dispositivo, aunque tiene la desventaja que no se implementa de manera automática el chequeo de errores. Es por esto, que es importante validar la comunicación con este tipo de transferencia e implementar de manera manual el chequeo de errores.

Adicional a la transferencia isócrona, se debe implementar la transferencia de control, la cual es requerida por cualquier dispositivo USB para el proceso de enumeración y para la desconexión del dispositivo. Por último, también se considera implementar una transferencia de interrupción, ya que la adquisición de datos debe de controlarse desde la PC, es decir el usuario selecciona cuando iniciar este proceso o cuando terminarlo.

Por parte del DSP se trabajó con estos tres tipos de transferencias, configurando los registros requeridos para establecer la comunicación y validando que la lectura de las señales y el envío de los datos no interfieran en el procesamiento del DSP.

Durante el presente trabajo, se configuró primeramente los endpoints, IN y OUT, correspondientes a la transferencia de control para validar que la PC pueda reconocer el dispositivo y se genere el proceso de inicialización y enumeración en el DSP. Prácticamente estos endpoints auxilian al DSP cuando el usuario conecta o desconecta el dispositivo de un puerto USB en la PC, con la finalidad que se generen los eventos requeridos para reconocer el dispositivo o para deshabilitar la comunicación.

En la figura 4.15, se muestra el código de una función la cual implementa una respuesta a los eventos del bus USB y maneja los paquetes de configuración del USB. Básicamente maneja los eventos y lógica de los endpoints correspondientes a transferencias de control.

Un primer problema encontrado con el uso de transferencia isochronous es que no existe un driver genérico para Windows que se pueda utilizar, para que una PC pueda reconocer al dispositivo USB. Por lo tanto en este punto se tuvo que trabajar también en el desarrollo de controladores para Windows, se usó para esto el Windows Driver Kit (WDK) de Microsoft para generar un driver específico del dispositivo utilizando transferencia isócrona.

Al finalizar de escribir el código del driver para comunicar el dispositivo USB con la PC, se realizan pruebas de validación con la finalidad de que la PC pueda reconocer correctamente el dispositivo conectado. A continuación se comienza a trabajar con el envío de datos previamente filtrados a la PC, mediante el puerto USB. Y con la transferencia de interrupción para iniciar y parar la adquisición de datos desde la PC.

```
void USB_ctl(USB_DevNum DevNum, USB_EpHandle hEp0In,USB_EpHandle hEp0Out)
{
    Uint16  Request;
    USB_REQUEST_RET ReqHandlerRet = USB_REQUEST_DONE;
    USB_EVENT_MASK  USB_ctl_events;


    // find out the endpt0 event caused this funcion to be called
    // and respond to the events
    USB_ctl_events = (USB_getEvents(hEp0Out) | USB_getEvents(hEp0In));

    // if the USB reset request received, abort all endpoint activities
    // and reconfigure the USB module

    if(USB_ctl_events & USB_EVENT_RESET)        // USB RESET event received
    {
        USB_abortAllTransaction(DevNum);        // stop all data transfer activities
        usbCurConfigStat = 0x00;                // reset device config number
        USB_init(DevNum, myUsbConfig, 0x80);      // reconfig the USB module
    }

    if (USB_ctl_events & USB_EVENT_SUSPEND)     // USB SUSPEND event received
    {
        USB_issueRemoteWakeup( DevNum );
    }


    // if the event is a setup packet received event then read the setup packet,
    // and lookup the USB_ReqTable[] for the appropriate request handler

    if((USB_ctl_events & USB_EVENT_SETUP) == USB_EVENT_SETUP)
    {
        // read the setup packet, if something wrong with setup packet then stall
        // the control endpints
        if(USB_getSetupPacket(DevNum, &USB_Setup) == USB_FALSE)
        {
            ReqHandlerRet = USB_REQUEST_STALL;
        }
        else
        {
            #if(0)
                *data++ = 0x1234;
                *data++ = USB_Setup.bmRequestType;
                *data++ = USB_Setup.bRequest;
                *data++ = USB_Setup.wValue;
                *data++ = USB_Setup.wIndex;
                *data++ = USB_Setup.wLength;
            #endif

            // lookup the USB request handler
            Request = ((USB_Setup.bmRequestType&0xC0)<<8)|USB_Setup.bRequest;
            fpRequestHandler = USB_lookupReqHandler(Request, USB_ReqTable);
        }

    }   // end of if setup event received
```

Figura 4.15 Código para manejar eventos en endpoints de control.

En la figura 4.16 se muestra una parte del código implementado de la rutina para realizar una transferencia isócrona en un dispositivo USB.

Finalmente se logró la comunicación correcta y el driver desarrollado funciona correctamente en la PC.

Se genera un código en LabWindows/CVI, importando la dll que se generó del driver para poder trabajar con las funciones de comunicación al dispositivo USB.

```
VOID PerformIsochTransfer(_In_ PDEVICE_CONTEXT DeviceContext, _In_ WDFREQUEST Request, _In_ ULONG  TotalLength)
{
    ULONG                       numberOfPackets;
    NTSTATUS                    status;
    PREQUEST_CONTEXT            rwContext;
    WDFUSBPIPE                  pipe;
    PFILE_CONTEXT               fileContext;
    WDF_OBJECT_ATTRIBUTES       attributes;
    ULONG                       j;
    USBD_PIPE_HANDLE            usbdPipeHandle;
    PMDL                        requestMdl;
    WDFMEMORY                   urbMemory;
    PURB                        urb;
    size_t                      urbSize;
    ULONG                       offset;
    ULONG                       frameNumber, numberOfFrames;
    PPIPE_CONTEXT               pipeContext;

    rwContext = GetRequestContext(Request);

    UsbSamp_DbgPrint(3, ("PerformIsochTransfer %s for Length %d - begins\n",
                         rwContext->Read ? "Read":"Write", TotalLength));
    //
    // Get the pipe associate with this request.
    //
    fileContext = GetFileContext(WdfRequestGetFileObject(Request));
    pipe = fileContext->Pipe;
    pipeContext = GetPipeContext(pipe);

    if ((TotalLength % pipeContext->TransferSizePerFrame) != 0) {
        UsbSamp_DbgPrint(1, ("The transfer must evenly start and end on whole frame boundaries.\n"));
        UsbSamp_DbgPrint(1, ("Transfer length should be multiples of %d\n", pipeContext->TransferSizePerFrame));
        status = STATUS_INVALID_PARAMETER;
        goto Exit;
    }

    if (DeviceContext->IsDeviceSuperSpeed) {

        numberOfFrames  = TotalLength / pipeContext->TransferSizePerFrame;
        numberOfPackets = TotalLength / pipeContext->TransferSizePerMicroframe;

        //
        // Then make sure the buffer doesn't exceed maximum allowed packets per transfer
        //

        if (numberOfPackets > MAX_SUPPORTED_PACKETS_FOR_SUPER_SPEED) {
            UsbSamp_DbgPrint(1, ("NumberOfPackets %d required to transfer exceeds the limit %d\n",
                                 numberOfPackets, MAX_SUPPORTED_PACKETS_FOR_SUPER_SPEED));
            status = STATUS_INVALID_PARAMETER;
            goto Exit;
        }

        UsbSamp_DbgPrint(3, ("Will send %d packets of %d bytes in %d frames\n",
                             numberOfPackets, pipeContext->TransferSizePerMicroframe, numberOfFrames));

    } else if (DeviceContext->IsDeviceHighSpeed) {
```

Figura 4.16 Código para transferencia isócrona en el DSP.

4.5 SOFTWARE PARA INTERFAZ DE USUARIO

El software original del EEG Digital desarrollado en CIDESI, fue desarrollado bajo la plataforma de LabWindows/CVI, por lo tanto para este trabajo se continuó bajo esta plataforma, básicamente se realizó ajustes en el programa original para reemplazar las etapas de adquisición y procesamientos de las señales.

En la versión original, la adquisición de datos se realizaba con el uso de una tarjeta de adquisición de datos comercial de la empresa National Instruments, bajo el puerto PCI de una PC, y en software se utilizan los drivers provistos por National Instruments para comunicación con sus tarjetas de adquisición de datos.

Aquí es donde se realizó el primer cambio a la aplicación, ya que ahora la adquisición de los datos se realiza en la tarjeta electrónica desarrollada y los datos se envían desde esta tarjeta hacia la PC mediante el uso del puerto de comunicación USB. Como ya se mencionó anteriormente, se tuvo que realizar el desarrollo de un driver para la comunicación entre la tarjeta desarrollada y la PC.

La nueva versión de código del software del EEG Digital utiliza este driver desarrollado y modifica el código de la adquisición de datos.

En la aplicación original del EEG, después de realizar la adquisición de datos se realiza el filtrado digital de las señales adquiridas, antes de mostrar las señales al usuario y de almacenarlas en la PC. Ahora con el desarrollo realizado en este trabajo, los datos enviados desde la tarjeta electrónica hacia la PC son ya datos con señales filtradas digitalmente por lo que otro cambio más a la aplicación original fue eliminar la etapa de filtrado en la aplicación de la PC.

Estos fueron los cambios realizados en la aplicación del EEG Digital, logrando finalmente tener una versión del EEG funcional con la tarjeta electrónica desarrollada en este trabajo.

A continuación se coloca la comparativa entre una adquisición utilizando el EEG como se tiene actualmente, figura 4.17, y la adquisición con el diseño desarrollado en este trabajo, figura 4.18.

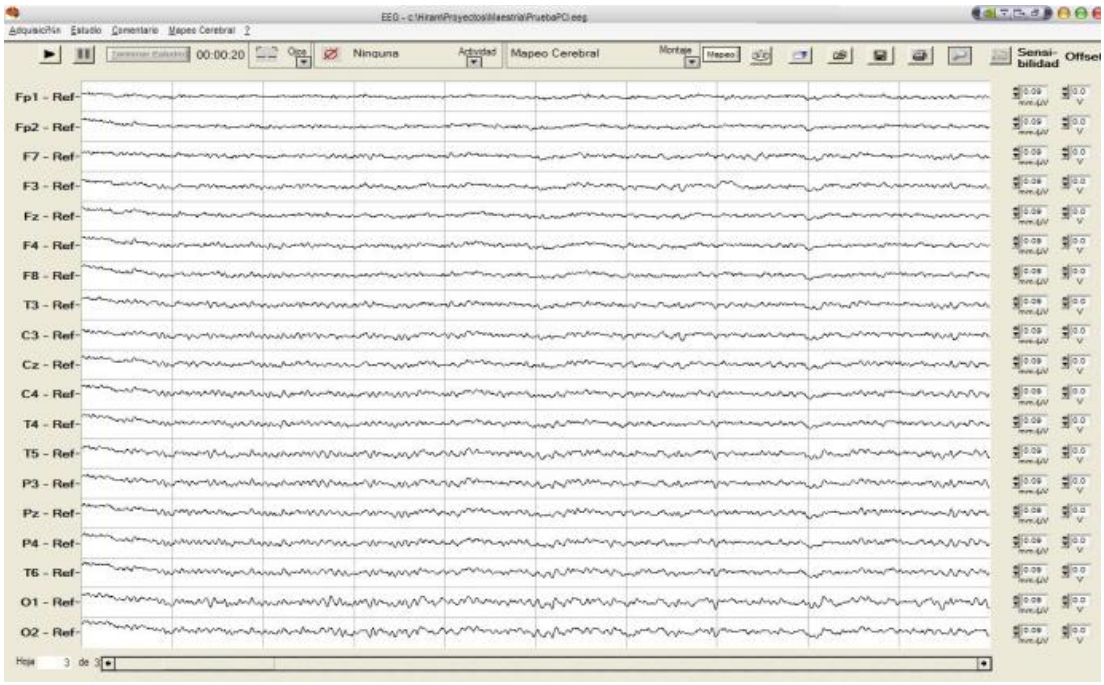


Figura 4.17 Resultado utilizando la tarjeta de adquisición de datos PCI comercial y una PC.

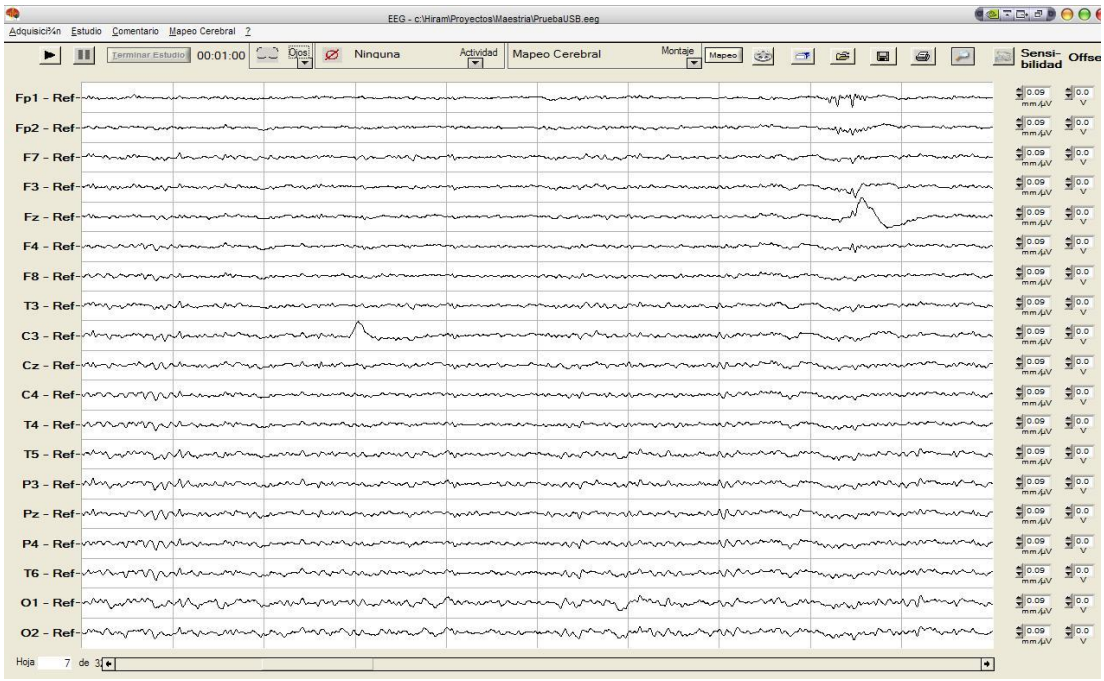


Figura 4.18 Resultado utilizando la tarjeta desarrollada en el presente trabajo y una PC.

CAPÍTULO V – CONCLUSIONES Y TRABAJO FUTURO

5.1 CONCLUSIONES

Finalmente se trabajó en el desarrollo de un prototipo del módulo de adquisición de datos, ya que todo el trabajo previo se realizó utilizando un módulo comercial de evaluación del DSP, pero este módulo es costoso, ya que contiene muchos componentes adicionales que para nuestra aplicación no son requeridos.

Por lo tanto a partir de analizar que componentes son requeridos, se procede a diseñar un prototipo que nos permita validar la adquisición, filtrado y comunicación USB, el cual se instala directamente en el amplificador del EEG Digital. En la figura 5.1 se muestra el prototipo desarrollado de manera final para este trabajo de tesis.

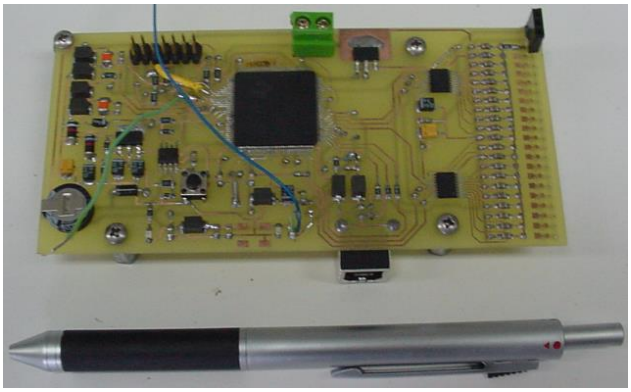


Figura 5.1 Prototipo electrónico final.

Se planteó inicialmente los siguientes objetivos:

- Desarrollo de software en un DSP para la adquisición de 21 señales analógicas, con una frecuencia de muestreo de hasta 1000Hz.
- Desarrollo del algoritmo para el filtrado digital de las señales en el propio DSP.
- Desarrollo de software para la comunicación entre el DSP y la PC mediante el puerto USB.
- Sincronizar la adquisición de datos con la comunicación USB.
- Modificar el software actual del EEG para integrar la comunicación con la tarjeta de adquisición de datos desarrollada en el proyecto.

Con el desarrollo completo en este trabajo se logró cumplir completamente con cada uno de estos objetivos específicos planteados al inicio del trabajo, vale la pena

comentar que en el tema de adquisición de datos se maneja el cambio de frecuencia de muestreo en valores predefinidos, debido a que se colocan los coeficientes para cada uno de los filtros en la memoria ROM del módulo desarrollado y no podría trabajar con cualquier frecuencia seleccionada en el rango requerido. Finalmente los valores que el usuario podría seleccionar para la frecuencia de muestreo en la adquisición de datos son: 200Hz, 500Hz y 1000Hz.

En la figura 5.2, se muestra el desarrollo completo realizado en el presente trabajo.

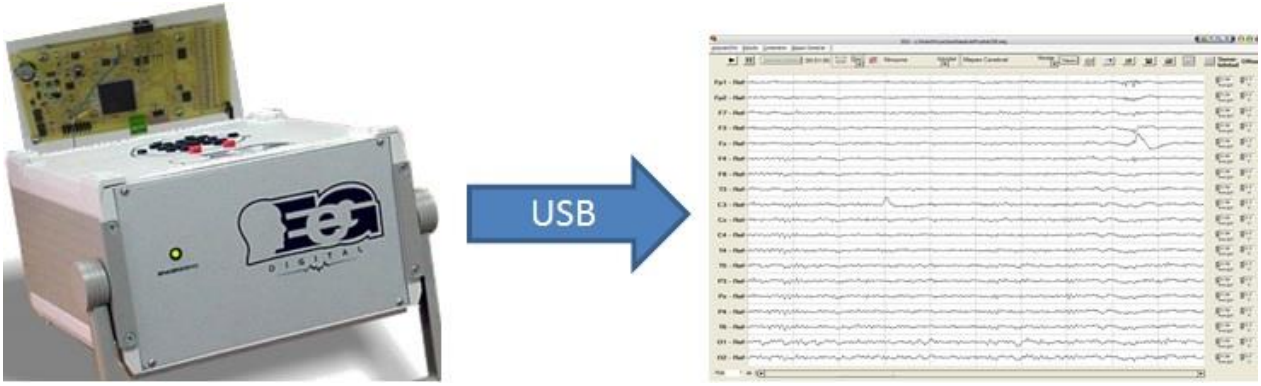


Figura 5.2 Resultado final del trabajo realizado.

5.2 TRABAJO FUTURO

Con el presente trabajo desarrollado se deja planteado para un trabajo futuro el continuar con este desarrollo para implementarlo bajo la especificación USB 3.0, para lo cual se requiere realizar el cambio del DSP por uno que soporte esta especificación.

Se plantea también un trabajo a futuro para realizar mayor procesamiento digital a lo que se ha implementado actualmente, como ya se mencionó el trabajo de procesamiento digital actual recae únicamente en la implementación de filtros digitales, pero con el poder de procesamiento de un DSP o de dispositivos más actuales como son los procesadores de aplicaciones multimedia (OMAP) ó procesadores escalables basados en ARM (Sitara), ambos dispositivos de la empresa Texas Instruments.

REFERENCIAS

[1] D. Lederman, F. W. Maloney, y L. Servén, *Lessons from NAFTA for Latin American and Caribbean (LAC) Countries: A Summary of Research Findings*, 2003.

[2] Secretaría de Salud México, *Programa de Acción: Investigación en Salud*, México DF, 2001.

[3] E. Biegeleisen, K. Nepal, T. Ning, *Real-time Analysis of Biomedical Signals using a High Speed DSP Board*, 2002.

[4] D. Cvetkovic, E. D. Übeyli, I. Cosic, *Wavelet transform feature extraction from human PPG, ECG, and EEG signal responses to ELF PEMF exposures: A pilot study*, Digital Signal Processing 18, pp. 861–874, 2007.

[5] M. Engin, et al, *A prototype portable system for EEG measurements*, Measurement 40, Elseiver Ltd, pp.936-942, 2007

[6] C. Guger, *Design of an EEG-based Brain-Computer Interface (BCI) from standard components running in real-time under Windows*, Biomed Tech (Berl), 44(1-2), pp.6-12, Jan-Feb 1999.

[7] R. Hornero, et al, *A DSP Implementation of Wavelet Transform to Detect Epileptiform Activity in the EEG*, Proceedings of the Eighth Annual International Conference on Signal Processing Applications and Technology, 2000.

[8] http://www.usb.org/developers

[9] A. Furfaro, et al, *Procesamiento Intensivo del ECG con procesadores IA-32 e IA-64*, XV Congreso Argentino de Bioingenieria, 2005.

[10] M. Najim, Digital Filters Design for Signal and Image Processing, iSTE, 2006.

[11] L. Sousa, at al, Generic Architecture Designed for Biomedical Embedded Systems, IFIP International Federation for Information Processing, Volume 231, Embedded System Design: Topics, pp. 353–362, 2007.

[12] B. Hollý, A. S. Hassan, *AD Converters and DSP in Biomedical Engineering Education*, 2003.

[13] American Academy of Neurology, *Assessment of Digital EEG, Quantitative EEG, and EEG Brain Mapping*, Report of the American Academy of Neurology and the American Clinical Neurophysiology Society, 1997.

[14] J.D. Bronzino, *The Biomedical Engineering Handbook*, CRC Press LLC, Second Edition, 2000.

[15] S. Meneses, *Acondicionamiento para Electroencefalógrafo*, Informe Técnico, 2004.

[16] T. J. Willis, *Biomedical Digital Signal Processing*, Prentice Hall, 2000.

[17] D. J. Mcfarland, A. T. Lefkowicz, J. R. Wolpaw, *Design and operation of an EEG-based brain–computer interface with digital signal processing technology*, Behavior Research Methods, Instruments, & Computers, 29 (3), pp.337–345, 1997

[18] S. W. Smith, *Digital Signal Processing - A Practical Guide for Engineers and Scientists*, Newnes, 2003.

[19] J. Bai, et al, *A Portable ECG and Blood Pressure Telemonitoring System,* IEEE Engineering in Medicine and Biology, pp. 63-70, 1999.

[20] M. S. Filho, H. A. Schneebeli, A. C. Machado, *An integrated software platform for the design and DSP-based implementation of digital filters*, IEEE, 1997.

[21] E. Jovanov, et al, *Real Time Holter Monitoring of Biomedical Signals*, DSP Technology and Education Conference DSPS, August 4-6, 1999, Houston, Texas.

[22] C. S. Ho, et al, *Design of Portable ECG Recorder with USB Storage*, IEEE, 2007

[23] D. Balasubramaniam, D. Nedumaran, *Doppler Spectrogram Calculation using CFFT Algorithm in a Digital Signal Processor Based System*, Third Asia International Conference on Modelling & Simulation, pp. 341-344, 2009.

[24] I. Bloch, *Information Fusion in Signal and Image Processing: Major Probabilistic and Non-probabilistic Numerical Approaches, Wiley, 2008.*

[25] W. Hamadene, L. Peyrodie, C. Vasseur, *Exploring the Nonlinear Dynamics of EEG Signals*, IEEE CCECE/CCGEI, Saskatoon, May 2005.

[26] M. Mesbah, B. Boashash, *Performance comparison of seizure detection methods using EEG of newborns for implementation of a DSP subsystem*, 2002.

[27] E. Jovanov, et al, *Real Time Portable Heart Monitoring Using Low Power DSP*, International Conference on Signal Processing Applications and Technology ICSPAT, 2000, Dallas, Texas.

[28] J. G. Proakis, Dimitris G. Manolakis, *Tratamiento digital de señales. Principios, algoritmos y aplicaciones*, Ed. Prentice Hall, 1998.

[29] J. Van de Vegte, *Fundamentals of Digital Signal Processing*, Ed. Prentice Hall, 2001.

[30] V. K. Madisetti, *Digital Signal Processing Handbook: Video, Speech and Audio Signal Processing and Associated Standards*, CRC Press, Second Edition, 2010.

[31] J. L. Semmlow, *Biosignal and Biomedical Image Processing MATLAB based Applications*, Marcel Dekker Inc, New York, NY, 2004.

[32] D. Stranneby, *Digital Signal Processing: DSP & Applications*, Newnes, 2001.

[33] Hu, Yu Hen, *Programmable Digital Signal Processors: Architecture, Programming and Applications*, New York, NY, USA: Marcel Dekker Incorporeted, 2001.

[34] G. Frantz, *DSP's Past Can't Hold a Candle to its Future*, Texas Instruments, 2007

[35] P. Lapsley and Jeff Bier, *DSP Processor Fundamentals Arquitectures and Features*, IEEE, 1997.

[36] S. M. Kuo, W. Seng Gan, *Digital Signal Processors Architectures, Implementations, and Applications*, Ed. Prentice Hall, 2004.

[37] S. Note, P. van Lierop, J. van Ginderdeuren*, Rapid Prototyping of DSP Systems: Requirements and Solutions*, 1995

[38] Compaq, Intel, Microsoft, NEC, *Universal Serial Bus Specification Rev. 1.1*, 1998.

[39] S. Olsen, *USB - Under the Hood and Looking Forward: An Introduction to USB and the future of USB*, 2007.

[40] B. Bo, S. Shuying, W. Chunping, *Design of Data Acquisition Equipment Based on USB*, 2007.

[41] C. Cansaya Herrera, *Desarrollo de una interfaz USB para el control de estaciones de radio HF y VHF para comunicación de datos*, 2005.

[42] W. Chengru, L.Yingwei, *The principle of USB 2.0 and project developing*, 2006.

[43] Z. Xi, *The introduction of USB technology*, 2005.

[44] Z. Nianhuai, J Hao, *The USB interface developing manual*, 2002.

[45] J. Hyde, *USB Design by Example: A Practical Guide to Building I/O Devices*, 2nd Edition, Intel Press, 2001.

[46] A. Jan, *USB Complete*, 4th Edition, Lakeview Research LLC, Madison, WI, 2009.

[47] R. Posada, et al, *USB Bulk Transfers between a PC and a PIC Microcontroller for Embedded Applications*, Electronics, Robotics and Automotive Mechanics Conference, 2008.

[48] Texas Instruments, *TLV2556 Datasheet*, Rev. 1, 2002.

[49] Spectrum Digital Inc, *TMS320VC5509 Evaluation Module Technical Reference*, Rev. B, 2001.

[50] Texas Instruments, *TMS320C5000 DSP/BIOS API Reference Guide*, 2003

[51] Texas Instruments, *TMS320C55x Chip Support Library API Reference Guide*, September 2004.

[52] Texas Instruments, *DSP/BIOS Driver Developer's Guide*, 2002.

[53] http://www.physionet.org/

ANEXO A

CIINDET 2008

6º Congreso Internacional en Innovación y Desarrollo Tecnológico,
8 al 10 de octubre de 2008, Cuernavaca, Morelos., México.

Diseño e implementación de un sistema electrónico para adquisición y comunicación de datos utilizando el puerto USB.

H. A. Hernández Rivera[1], S. Arciniega Montiel[1], J. C. Pedraza Ortega[2,] E. Gorrostieta Hurtado[2].

Resumen: El desarrollo de sistemas electrónicos utilizando procesadores de señales digitales ha ido aumentando paulatinamente, debido a la necesidad de manejar y procesar grandes cantidades de datos en tiempo real. Un ejemplo del uso de estos sistemas se expondrá en el presente trabajo, el cuál será implementado para el procesamiento de señales bioeléctricas. El desarrollo consiste en un algoritmo para adquirir, filtrar y enviar datos, hacia una computadora personal desde un procesador de señales digitales. La comunicación entre el procesador de señales digitales y la computadora se efectúa mediante el puerto serial universal, configurado para transferencias isócronas. El trabajo aquí desarrollado es aplicado para la adquisición y comunicación entre un Electroencefalógrafo Digital y una computadora. Es importante señalar que el algoritmo queda abierto para usarlo en cualquier otro dispositivo que requiera de una comunicación hacia una computadora y en los cuales se tenga como requerimiento crítico el tiempo de envío de los datos. Como resultado de este trabajo, se ha podido sustituir una tarjeta de adquisición de datos comercial. Logrando con esto el aumento de competitividad y reducción de costos entre equipos médicos desarrollados en el país y aquellos desarrollados en el extranjero.
Palabras Clave: Procesador de señales digitales, Algoritmo, Tiempo real, Puerto serial universal, Señales bioeléctricas, Electroencefalógrafo digital, Desarrollo de equipos médicos.
Abstract: Development of electronic systems using Digital Signal Processors has been increasing gradually, due to necessity to handle and to process a lot of data in real time. An example of that will be exposed on this work, which will be implemented for processing bioelectrical signals. The development consists of an algorithm to collect, filter and send data, towards a personal computer from a digital signal processor. The communication between digital signal processor and the computer uses the universal serial port, setting for isochronous transfer. The work developed here is applied for the acquisition and communication between a Digital Electroencephalogram and a computer. It is very important to indicate that the algorithm could be used in any other device that requires of a communication towards a computer and in which has critical requirement like the time of data sending. The result of this work, it has been possible to replace a commercial board of data acquisition. We can increase of competitiveness and reduction of costs between medical equipment developed in the country and those developed abroad.
Keywords: Digital signal processor, Algorithm, Real time, Universal serial port, Bioelectrical signals, Digital electroencephalogram, Develop of medical equipment.

Introducción

Históricamente México ha sido considerado como un país principalmente de mano de obra para la industria, no así como un país desarrollador de tecnología, se ha tenido que importar la mayor parte de la tecnología actualmente utilizada, tanto en la industria médica como en cualquier otra industria. Con el paso de los años, esto ha ido cambiando, debido a que se ha empezado a trabajar en conjunto con la investigación, para así alcanzar el objetivo propuesto, reducir la exportación de productos [1].

Hablando de la industria médica hasta hace aproximadamente unos 5 años había en el país muy pocas empresas las cuales se dedicaban al desarrollo de equipos médicos de diagnóstico en México, esto implica que se tuviera que importar cada equipo médico y logrando así los altos costos que tenían dichos equipos, por lo cual la secretaría de salud y hospitales

[1]Hiram Abif Hernández Rivera, Centro de Ingeniería y Desarrollo Industrial, Playa Pie de la Cuesta #702, Col. Desarrollo San Pablo, C.P.76130 Querétaro, Querétaro, México. ahernandez@cidesi.mx
[2]Sadot Arciniega Montiel. sadot@cidesi.mx
[2]Jesús Carlos Pedraza Ortega, Facultad de Informática Universidad Autónoma de Querétaro, Campus Juriquilla, Querétaro, Querétaro, México. caryoko@yahoo.com
[2]Efrén Gorrostieta Hurtado.

privados han invertido una fuerte cantidad de dinero para conseguir los aparatos necesarios para su infraestructura actual.

Así fue como en el 2003 el Centro de Ingeniería y Desarrollo Industrial (CIDESI) comenzó a trabajar en el desarrollo de equipos médicos, enfocándose en el área de neurología y actualmente en el área de cardiología.

El presente trabajo contribuye al desarrollo de equipos médicos de diagnóstico en el país, ya que será aplicado hacia un electroencefalógrafo (EEG) digital desarrollado por personal del CIDESI, cabe mencionar que éste es uno de los primeros equipos médicos de diagnóstico desarrollados completamente en nuestro país.

Hasta estos momentos el EEG digital ha utilizado una tarjeta de adquisición de datos comercial conectada a un puerto PCI de una PC para la adquisición de las señales bioeléctricas provenientes de un sistema de acondicionamiento, pero ahora se pretende remplazar la tarjeta de adquisición de datos comercial por una tarjeta desarrollada en CIDESI y que utiliza al puerto USB como medio de comunicación entre el EEG Digital y una computadora. Este cambio impactará directamente en el costo del producto, y a su vez se logra una conectividad hacia cualquier dispositivo USB, específicamente Laptops y PC´s, para así obtener una mayor portabilidad en el producto.

La mayoría de los trabajos desarrollados con comunicación USB, se basan en una comunicación *bulk*, en la cual no se requiere de tiempos precisos para la transferencia de datos, tal como mouse, teclado, disco duro, etc. Por otro lado, el trabajo aquí desarrollado se plantea el uso de una transferencia isócrona, la cual esta enfocada para transferencias de datos en donde se requiera de una latencia mínima en la comunicación.

Una de las aplicaciones potenciales de envío y procesamiento de información con requerimiento de latencia mínima y en la cuál los tiempos de transferencia y procesamiento son críticos es un electroencefalógrafo. Un electroencefalógrafo es un equipo médico, el cuál se encarga de capturar las señales eléctricas que genera el cerebro y procesarlas para el monitoreo en línea durante el estudio o almacenarlas para un análisis posterior de la señal. Estos equipos han sido ampliamente utilizados desde hace muchos años, para la detección de enfermedades como epilepsia, migraña, trastornos de sueño entre otros.

El EEG desarrollado en CIDESI cuenta con 21 electrodos monopolares y 2 electrodos de tierra, es capaz de adquirir las señales con una frecuencia de muestreo hasta de 1000 Hz por cada canal, cuenta con las herramientas de análisis como Mapeo cerebral y video sincronizado con los trazos del EEG.

La interfaz USB es lo suficientemente versátil para ser usada en una amplia cantidad de aplicaciones, tales como dispositivos periféricos [2]. Los dispositivos más utilizados con este tipo de comunicación son: mouse, teclados, almacenamiento de datos, impresoras y dispositivos de audio y video. Pero el protocolo también puede ser usado para unidades de adquisición de datos y en general para cualquier dispositivo que necesite de transferir o recibir información de algún otro.

El protocolo USB utiliza cuatro tipos de transferencias para enviar o recibir datos, las cuales son las siguientes [3]:

- Control.- Esta transferencia sólo es utilizada para realizar funciones definidas por la especificación del protocolo, es utilizada para leer información del dispositivo y seleccionar ciertas configuraciones del mismo. Todos los dispositivos deberán contar con este tipo de transferencia.
- *Bulk*.- Esta transferencia es utilizada para situaciones en las cuales la taza de transferencia de datos no es crítica, aquí las transferencias son rápidas pero si el bus se encuentra ocupado entonces los datos esperarán a que se desocupe y pueda atender la petición de la transferencia.
- Interrupción.- Son utilizadas en dispositivos que deben de recibir atención periódica de otro dispositivo o host. Junto con las transferencias de control son las únicas formas con las que un dispositivo de baja velocidad puede transferir datos.
- Isócrona.- Tienen el tiempo de entrega garantizado, pero no corrigen algún error en la comunicación. Es requerida en comunicación en tiempo real.

Desarrollo

El presente trabajo se realizó en base a una tarjeta de evaluación del procesador de señales digitales C5509 de la empresa Texas Instruments, el cual cuanta con el puerto USB como uno de sus periféricos, además se utilizaron 2 convertidores de señal analógica a digital de 12 canales cada uno, con lo cual obtenemos los 21 canales que se necesitan monitorear. Este es el

hardware total que se utiliza para este desarrollo, además del EEG Digital. En el presente trabajo también se realiza el desarrollo del algoritmo tanto para el DSP como para la PC.

Hardware.

El hardware utilizado en este trabajo consiste principalmente en una tarjeta de evaluación de un DSP, y 2 convertidores análogo-digital, TLV2556, utilizando el protocolo SPI para la comunicación entre el DSP y los convertidores análogo-digital.

Por lo cual nos enfocaremos en el uso del DSP de la familia 5000 de Texas Instruments, concretamente se trabajó con la tarjeta de evaluación TMS320VC5509EVM de la empresa Spectrum Digital, la cual funciona como un punto inicial del proyecto, para más adelante realizar la tarjeta propietaria de CIDESI.

tarjeta de evaluación (EVM) mencionada nos permite examinar las características del procesador de señales digitales C5509, para determinar si estas empatan con los requerimientos de la aplicación deseada. Además, el módulo es una excelente plataforma para desarrollar y ejecutar software para los procesadores de señales digitales TMS320VC5509 [4].

Algunas de las características con las que cuenta la tarjeta de evaluación son las siguientes, y en la **Figura 1** se muestra el diagrama a bloques:

- VC5509A operando a 120 MHz.
- Interfaz de Bus Serial Universal (USB) 1.1.
- Controlador de sonido estéreo TLV320AIC23 con conectores para entrada de audio y salida para audífonos.
- Display de cristal líquido (LCD) 128x64 y teclado con 9 botones y potenciómetros.
- 4 conectores de expansión (datos, entradas/salidas, control e interfaz de host port).
- Interfaz para memorias tipo Memory Stick y Media Card.
- Memoria RAM 4Meg x 16 DRAM.
- Memoria Flash 1 Meg x 16.
- Conector JTAG (1149.1) para emulación.
- Operación a 5 Volts CD.

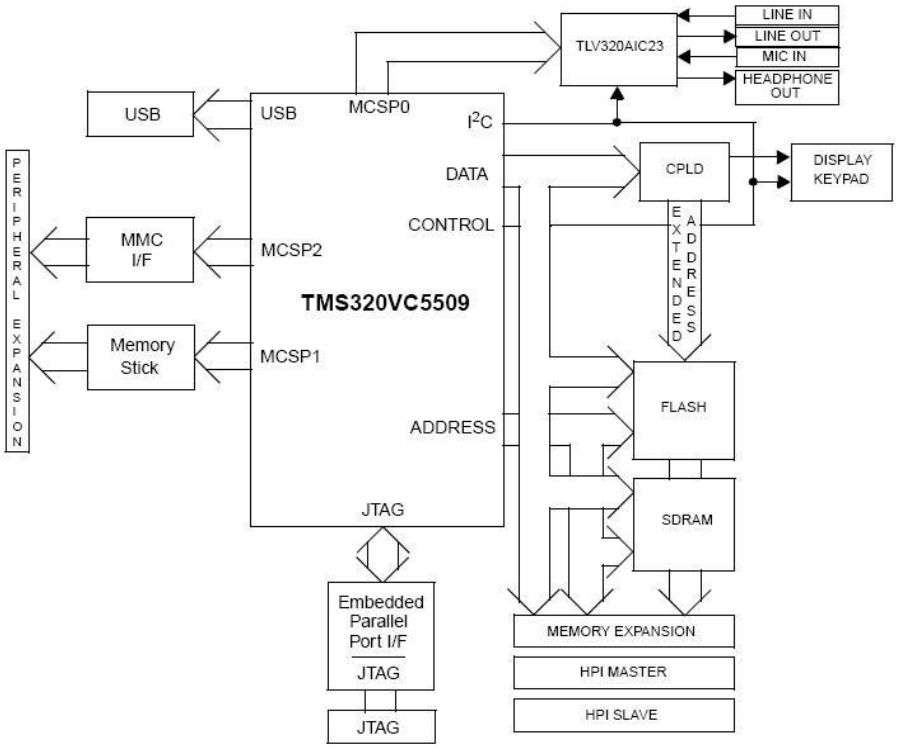


Fig. 1 Diagrama a bloques de la tarjeta de evaluación TMS320VC5509.

El procesador digital de señales C5509 cuenta con un puerto de comunicación USB con el cual es posible desarrollar dispositivos USB *full speed*, el cual es completamente compatible con la versión 1.1 de la especificación del bus serial universal (USB) con una taza de transferencia de datos de hasta 12 Mb/s [5].

El dispositivo cuenta con 16 puntos de acceso a la comunicación llamados *endpoints*, dos son especialmente para transferencias de control, de los cuales uno es para entrada y el segundo para salida. Y catorce *endpoints* de propósito general, donde 7 son de entrada y 7 salida, con estos 14 *endpoints* se pueden establecer cualquiera de las transferencias restantes, *bulk*, interrupción e isócrona [6].

En la **Figura 2** se muestra de manera conceptual cada uno de los bloques del módulo USB presentes en el DSP C5509, los bloques sombreados no son parte del módulo USB pero si del DSP. El bloque denominado SIE es el manejador del protocolo de USB. Así pues en esta figura nos muestra la interacción que existe entre los diferentes bloques del módulo de USB con el CPU y la memoria del DSP para llevar a cabo transferencias tanto de entrada como de salida de datos vía el puerto USB.

Es ampliamente recomendable el uso del controlador DMA, para transferir datos entre la memoria del DSP y el módulo de USB, para así reducir el consumo de tareas al procesador, debido a que las transferencias por DMA son mucho más efectivas, que transferir directamente los datos de la memoria del DSP sin el uso de DMA.

Algunas características del circuito integrado TLV2556, utilizado para la adquisición de los datos, son [7]:

- 12 Bits de resolución del convertidor.
- Hasta 200 kSps
- 11 canales de entrada analógica.
- Máximo error de linealidad ±1 LSB.
- Longitud de datos de salida programables
- Interfaz serial SPI hasta 15MHz

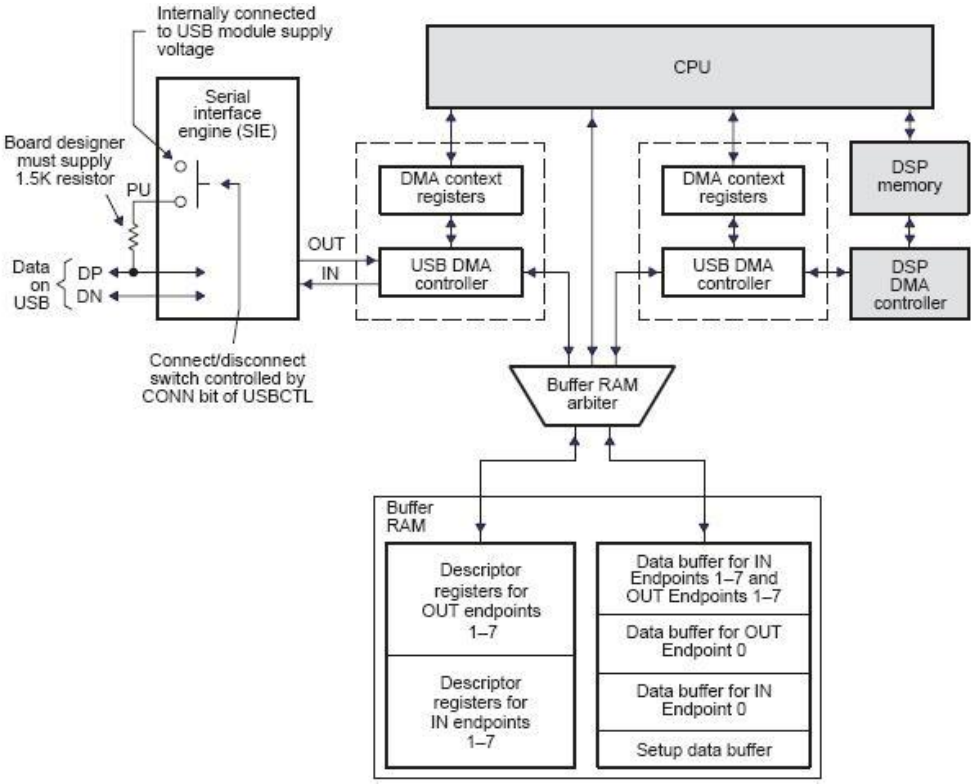


Fig. 2 Diagrama a bloques del módulo USB en el DSP C5509A.

Software y Firmware.

El software de la PC para la aplicación del EEG fue realizado en lenguaje C, bajo la plataforma de LabWindows/CVI, a la cuál se le incorporó una librería de funciones para realizar la comunicación usb.

El firmware del DSP fue realizado igualmente en lenguaje C, pero bajo la plataforma de Code Composer Studio, la cuál es dedicada para la programación de los DSP's de la empresa Texas Instruments. A continuación se mencionarán las características del firmware desarrollado para la aplicación.

Debido a que la aplicación requiere adquirir, filtrar y enviar los datos hacia una PC donde el tiempo de envío se torna un punto crítico en la aplicación, se decidió usar una herramienta muy poderosa de los DSP's de Texas Instruments la cual es llamada DSP/BIOS. Esta herramienta es un kernel de tiempo real escalable, la cual nos provee de herramientas para sincronizar y calendarizar tareas críticas en forma paralela.

En la **Figura 3** se puede visualizar el diagrama de estados del software desarrollado en el DSP.

Por características naturales de las señales bioeléctricas, éstas se adquieren con un nivel de ruido, por lo que se necesita realizar el filtrado de cada una de las señales adquiridas, así pues, una vez que se tiene listo un grupo de adquisiciones de todos los canales se procede a realizar el filtrado de las mismas. Existen diferentes tipos de filtros digitales que se pueden implementar en un DSP, entre los que podemos destacar, son los siguientes: Respuesta Finita al Impulso (FIR), Respuesta Infinita al Impulso (IIR), filtros adaptivos y filtros basados en *wavelets.*

En nuestro caso se trabajó con un filtro FIR con ventana Hamming pasa bandas con un rechaza banda a 60 Hz. La ecuación (1) es la que representa a un filtro FIR, donde h_i son los coeficientes del filtro,

$$y(n) = \sum_{i=0}^{M-1} h_i x(n-i) \qquad (1)$$

Antes de aplicar el filtro se debe proceder a calcular cada uno de los coeficientes, los cuales son almacenados en un vector de una longitud definida, la cual representa el orden del filtro a aplicar [8].

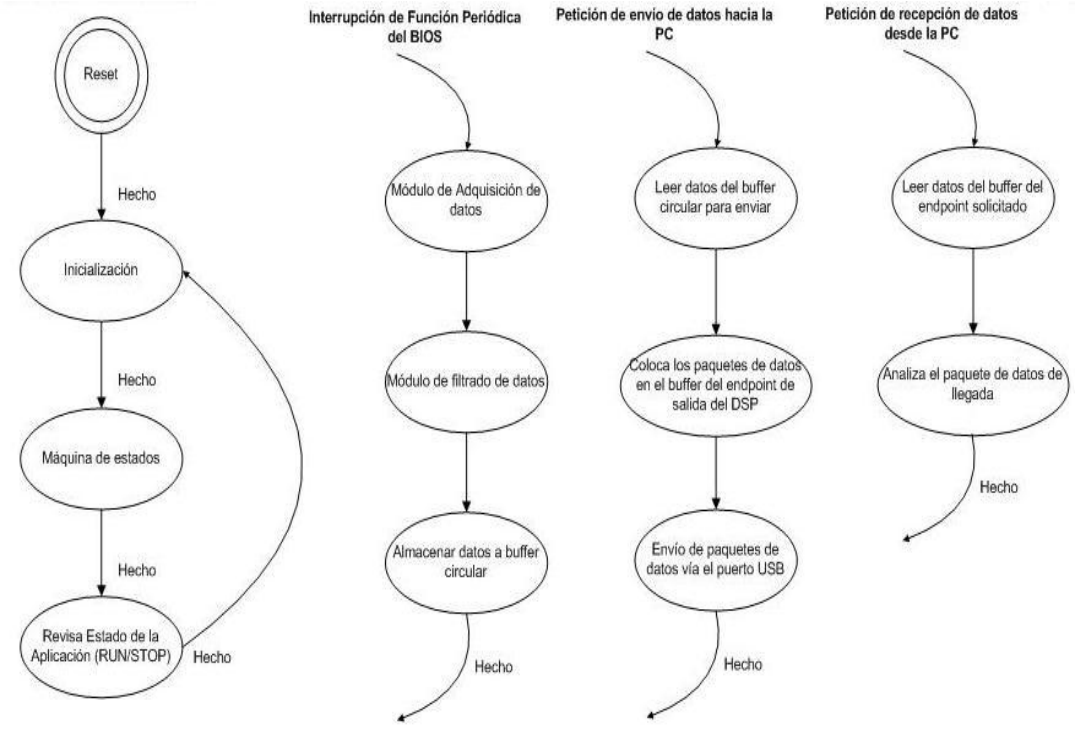


Fig. 3 Diagrama de estados del software en el DSP.

Para realizar el cálculo de los coeficientes se definen las siguientes ecuaciones.

La ecuación (2) nos representa la respuesta al impulso del filtro idealmente, llamada función *sinc*, si se realiza la convolución mencionada en la ecuación nos genera un filtro pasa bajos perfecto, el problema es que (2) es una función continua y por lo tanto nos genera coeficientes positivos y negativos indefinidamente, lo cual es complicado realizar en un DSP.

$$h[i] = \frac{sen(2\pi f_c i)}{i\pi} \quad (2)$$

A continuación se coloca la ecuación representativa de la ventana Hamming (3), la cuál se ejecuta desde i=0 hasta i=M, para un total de M+1 puntos.

$$w[i] = 0.54 - 0.46\cos(2\pi i / M) \quad (3)$$

Después debemos cambiar (2) por (4), la cual ya fue agregada la ventana Hamming.

$$h[i] = K \frac{sen(2\pi f_c (i - M/2))}{i - M/2} [0.54 - 0.46\cos(2\pi i / M \quad (4)$$

La ecuación (4) nos proporciona los coeficientes a utilizar en el filtro digital, una vez que se tienen almacenados los coeficientes, se aplica el filtro digital utilizando (1).

Cuando se tienen los datos filtrados, éstos se almacenan en un buffer circular debido a que no es recomendable que por cada adquisición se envíen datos hacia la PC, así que se espera a obtener un grupo de datos, para así enviar todos de una sola vez y tener una mejor comunicación.

Resultados

Se realizaron diferentes pruebas comparativas entre el uso de la tarjeta de adquisición de datos comercial contra el trabajo desarrollado aquí, con las cuáles pudimos obtener señales similares en ambas plataformas, por lo que podemos asumir que el reemplazo de la tarjeta comercial será un hecho para el producto propuesto.

En la **Figura 4** se muestra el entorno de desarrollo del software para el DSP, en el cual se muestra una parte del código de adquisición y filtrado de los datos de la tarjeta utilizada junto con el envío de datos por USB.

Conclusiones.

Como ya se mencionó anteriormente con el trabajo presentado se obtuvieron resultados con los cuales se puede remplazar la actual plataforma con la que se comunica el EEG Digital de CIDESI hacia una PC con la desarrollada aquí.

Es importante notar que el algoritmo fue propuesto para poder ser aplicado a futuros trabajos con pequeñas modificaciones obteniendo el mismo funcionamiento.

Una vez que ya tenemos validado el uso del DSP para la adquisición, procesamiento y envío de datos hacia una PC, se tiene planeado el uso de otro dispositivo más poderoso aún que el DSP, el cual es un OMAP, que cuenta con un DSP y un microcontrolador en su interior, el cual cuenta con interfaces como usb, bluetooth, wi-fi, entre otros. Este último dispositivo es el más actual con el que cuenta la empresa Texas Instruments. Además se pretende realizar el filtrado de las señales utilizando *wavelets*. El uso de este dispositivo se plantea debido a que se pretende que la herramienta de sincronización de video, con la cual cuenta el EEG Digital desarrollado en CIDESI, se implemente dentro del OMAP y no así en la PC como hasta ahora ha sido, además de las ventajas de comunicación, debido a que cuenta con una mayor cantidad de interfaces que el DSP.

Y como se ha comentado el algoritmo desarrollado nos permite utilizarlo para diferentes aplicaciones, otra de las aplicaciones donde se útilizará es en un Electrocardiógrafo de 12 canales inalámbrico para 5 pacientes, en su módulo concentrador.

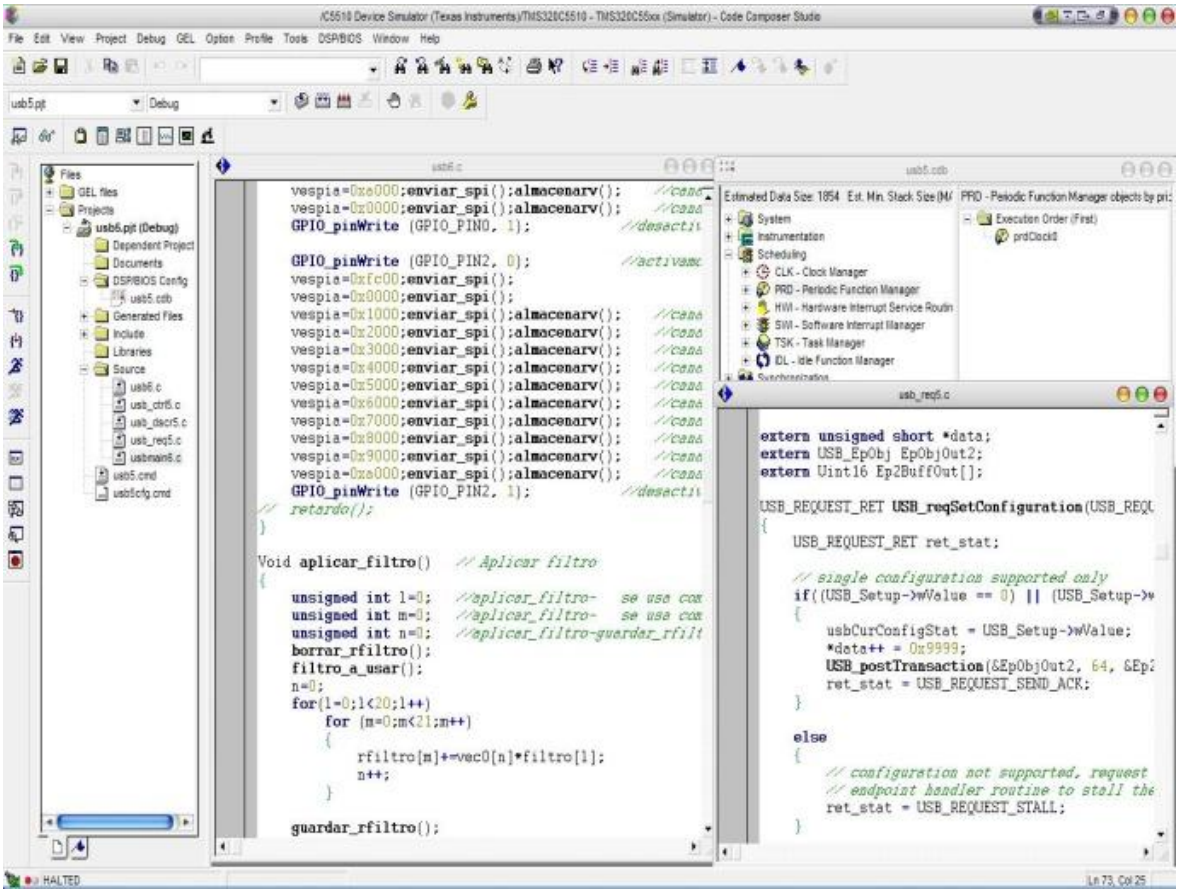


Fig. 4 Interfaz de desarrollo para el firmware del DSP.

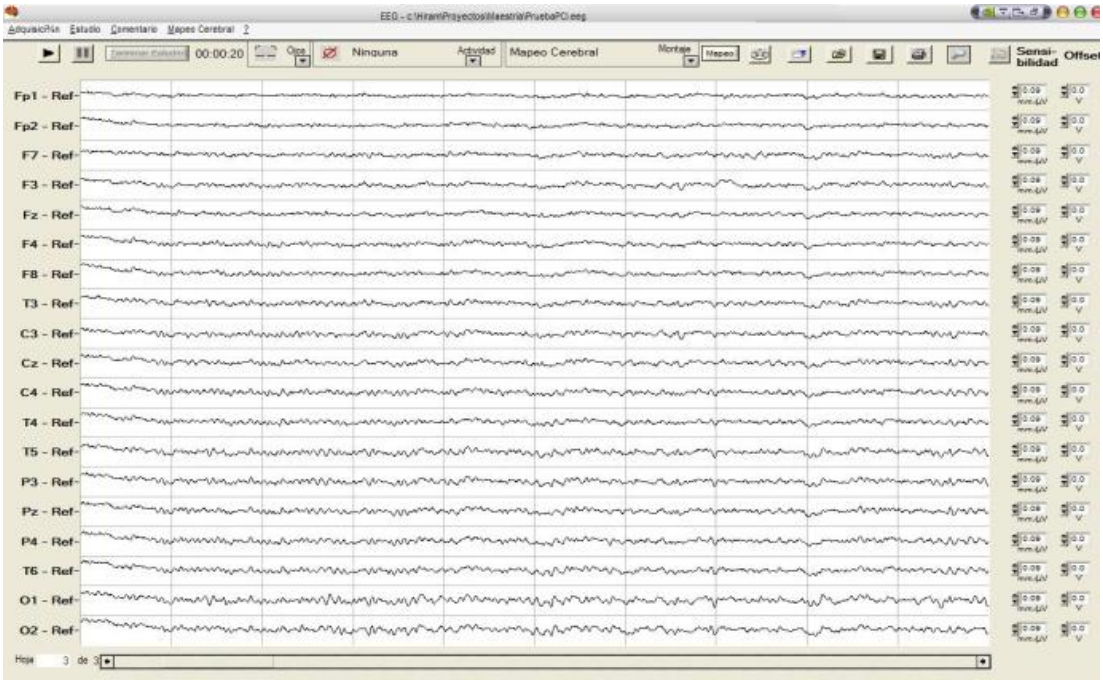


Fig. 5 Resultado utilizando la tarjeta de adquisición de datos PCI comercial y una PC.

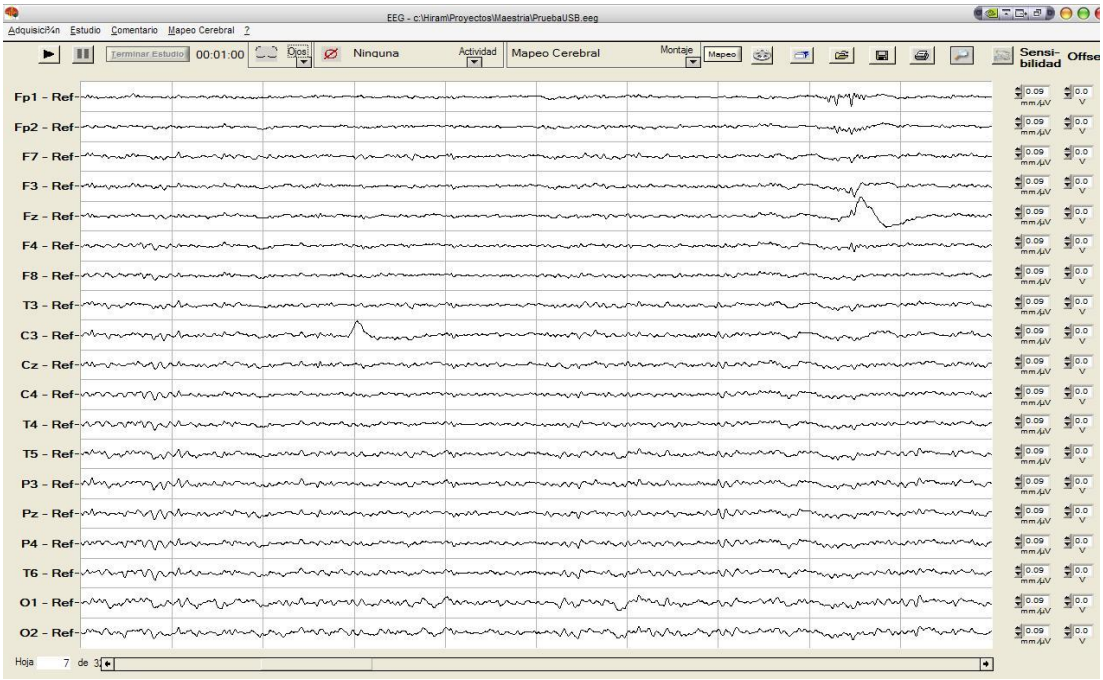


Fig. 6 Resultado utilizando la tarjeta con comunicación USB y una PC.

Referencias.

[1] Lederman Daniel, Maloney F. William y Servén Luis, *Lessons from NAFTA for Latin American and Caribbean (LAC) Countries: A Summary of Research Findings*, 2003.
[2] Hyde John, *USB Design by Example: A Practical Guide to Building I/O Devices, USA:* Intel Press, 2001.
[3] Axelson Jan, *USB Complete: Everything You Need to Develop Custom USB Peripherals*, Madison: Lakeview Research LLC, 2005.
[4] Spectrum Digital Inc, *TMS320VC5509 Evaluation Module Technical Reference*, 2001.
[5] Compaq, Intel, Microsoft, NEC, *Universal Serial Bus Specification Rev. 1.1*, 1998.
[6] Texas Instruments, *TMS320C55x DSP Peripherals Reference Guide*, 2001.
[7] Texas Instruments, *TLV2556 Data Sheet*, 2002.
[8] Steven W. Smith, *Digital Signal Processing: A Practical Guide for Engineers and Scientists*, Newnes, 2001.

Hiram Abif Hernández Rivera
Ingeniero en Electrónica egresado del Instituto Tecnológico de Querétaro en 2003. Ingeniero de proyectos en el área de Electrónica Aplicada, en CIDESI desde 2003. Actualmente Alumno de la Maestría en Mecatrónica en el Centro de Ingeniería y Desarrollo Industrial.

Sadot Arciniega Montiel
Ingeniero en Electronica egresado del Instituto Tecnológico de Querétaro en 1993. Obtuvo el grado de Maestría en Control e Instrumentación en 2003 en la Universidad Autónoma de Querétaro. Gerente del área de Electrónica Aplicada en CIDESI.

Jesús Carlos Pedraza Ortega
Ingeniero en Electrónica egresado del Instituto Tecnológico de Celaya en 1993. Cursó la Maestría en Ingeniería Eléctrica en la FIMEE, Universidad de Guanajuato. Sus estudios de Doctorado fueron en Ing. Mecánica en la Universidad de Tsukuba en Japón en 2002. Actualmente es Profesor-Investigador en la Facultad de Informática de la Universidad Autónoma de Querétaro.

Efrén Gorrostieta Hurtado
Ingeniero Electrónico egresado del Instituto Tecnológico y de Estudios Superiores de Occidente (ITESO). Cursó la Maestría y Doctorado en Mecatrónica en el Centro de Ingeniería y Desarrollo Industrial. Actualmente es Profesor-Investigador en la Facultad de Informática de la Universidad Autónoma de Querétaro