CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL

Unidad Zacatenco

Departamento de Computación

Desarrollo de un Software de Medición del Estrés para un Dispositivo Foto-pletismógrafo Basado en el Protocolo USB 2.0

Tesis que presenta

Andrés Bernal Jiménez

para obtener el Grado de

Maestro en Ciencias en Computación

Director de la Tesis

Dr. Adriano De Luca Pennacchia

México, D.F. Abril 2010

Resumen

El estudio de la medición de estrés es un reto para los médicos y psicólogos, que mediante diferentes métodos tratan de saber el nivel de estrés de un paciente. La importancia de dicha medición recae en que poder determinar si una persona está afectada por el estrés o no, por consecuencia esto permitiría definir un mejor tratamiento a aplicar en un paciente.

Estudios formales han contribuido en el establecimiento de las variables participantes en los síntomas del estrés, y con ello se han desarrollado mecanismos de medición que permiten su estudio. Los mecanismos usuales para llevar a cabo este tipo de mediciones es mediante pruebas fisiológicas y psicológicas.

En trabajos anteriores se ha mejorado la adquisición de los datos, así como el hardware del dispositivo necesario para la obtención de la señal fisiológica. El análisis de los datos debe ser mostrado gráficamente, posteriormente interpretado por personal clínico, psicólogos y personas interesadas en el estudio del estrés. La cantidad de muestras obtenidas es insuficiente para un análisis más detallado de la información, lo que produce resultados inexactos. De ahí la necesidad de implementar un nuevo esquema de adquisición de los datos provenientes de la parte del hardware y software.

Por lo tanto, el trabajo que aquí se presenta es un sistema que implementa un mecanismo de medición basado en mediciones fisiológicas. A este sistema de medición del estrés se le ha denominado *"Stress"*.

Corresponde a este trabajo, interpretar y presentar el resultado del análisis de los datos de una manera más eficiente al realizado anteriormente, así también aumentar la cantidad de muestras obtenidas por segundo mejorando la transmisión por medio del protocolo USB 2.0.

Nosotros utilizamos un esquema basado en el uso de la programación del lenguaje Java en combinación con el lenguaje C, permitiendo una mayor velocidad en la transmisión de los datos y facilitando la programación y el desarrollo de posteriores actualizaciones al sistema.

Abstract

Nowadays the stress measurement is a challenge for physicians and psychologists, that using different methods they try to find the stress level of patients. The importance of these measurements lies in order to determine whether a person is affected by stress or not, this result would define better treatment to apply in one patient.

There are formal studies that have helped to establish the variables involved in stress symptoms, and thus have developed measurement tools that allow its study. The usual mechanisms used for these measurements are physiological and psychological tests.

In previous works has improved the data acquisition and the hardware necessary for obtaining physiological signal. The analysis of data should be shown graphically and later interpreted by clinicians, psychologists and others interested in the study of stress. The number of samples is insufficient for a more detailed analysis of information, leading to inaccurate results. That is why the need arises to implement a new scheme of data acquisition from the hardware and software.

Therefore, we present a system which implements a measuring mechanism based on physiological measurements, that is called "Stress".

In this system we interpret and present the result of analysis of data more efficiently to previous made, and we increase the number of samples taken per second too, improving the transmission through the USB 2.0 protocol.

We use an scheme that is based on combination of Java and C programming languages, increasing the speed of data transmission and facilitating the programming and development of subsequent updates to the system.

Agradecimientos

A Dios, por brindarme el don de la vida y regalarme una familia maravillosa.

A mi madre querida: *A ti mi chaparrita, la mujer más maravillosa e importante en mi vida, gracias por todo ese amor que me has brindado, por tus acertados consejos, regaños y tu apoyo incondicional. Nunca serán suficientes las palabras para agradecerte, gracias, te quiero mucho.*

A mis hermanos:
A ti Paco, gracias por tu apoyo, compañia en momentos difíciles y por enseñarme a tener la fortaleza para superar cualquier obstáculo en esta vida. Y claro, gracias a mi hermanito Chuchin, por tu cariño, por recibirme siempre con una sonrisa y abrazo, los quiero mucho.

A mi familia:
Gracias a mi querida familia, por todo su cariño y todos esos momentos maravillosos que hemos compartido, a todos ellos gracias.

A mi asesor:
Dr. Adriano de Luca Pennacchia, gracias por su confianza que ha depositado en mí, por su paciencia y dedicación para dirigir este trabajo, y principalmente por sus enseñanzas.

A mis revisores:
Dra. Sonia Mendoza Chapa y Dr. José Guadalupe Rodríguez García: por sus acertadas correcciones y por el tiempo dedicado a la revisión de esta tesis.

A mis profesores:
A ellos que a lo largo de estos años de estudio, siempre fueron punto de apoyo para lograr terminar la maestría.

Agradezco al Dr. Abdiel Cáceres González por su confianza y por animarme a estudiar la maestría en el Cinvestav.

Agradezco al Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional por brindarme la formación y las herramientas suficientes durante mis estudios de maestría.

Gracias al Consejo Nacional de Ciencia y Tecnología por haberme proporcionado el apoyo económico para sustentar mi estancia durante el tiempo que realicé mis estudios de maestría.

Gracias al Consejo Mexiquense de Ciencia y Tecnología por el apoyo económico para sustentar los últimos meses para el termino de esta tesis.

Sofía Reza:
Gracias por tu apoyo, por tus sabios consejos, por hacer del Departamento de Computación un mejor lugar y porque cada mañana nos recibes con una calurosa sonrisa.

A mis amigos:
Agradezco a todos mis amigos de la maestría, que siempre estuvieron ahí para tenderme la mano, por la amistad que siempre me brindaron, por esos todos esos momentos que pasamos juntos estudiando y desvelándonos para entregar los trabajos a tiempo, por lar tardes interminables de quake, ping pong, ajedrez, etc. Les agradezco infinitamente, mencionarlos sería poco, aun así, gracias Ray, Gox, July, Lil, Ana, Pam, Pau, Jhony, Beto, Migue, Karman y como olvidar a mis amigos y queridos vecinos Christian y Madai, y los que me faltaron por mencionar igual muchas gracias.

Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1

Antecedentes, motivación, objetivos y metodología

1.1 Antecedentes

El estrés es la respuesta automática y natural de nuestro cuerpo ante situaciones normales de sobrevivencia incluidas las amenazadoras o desafiantes [27]. El entorno cambiante en que vivimos nos exige continuas adaptaciones, por lo tanto, cierta cantidad de estrés es necesaria [7].

Se tiende a creer que el estrés es consecuencia de circunstancias externas a nosotros, cuando en realidad, se ha podido entender que es un proceso de interacción entre los eventos del entorno y nuestras respuestas cognitivas, emocionales y físicas [8].

Su estudio inicia en 1867, cuando el fisiólogo francés Claude Bernard sugirió que los cambios externos en el ambiente podrían perturbar el organismo y que una de la principales características de los seres vivos es mantener la estabilidad de entre si mismos y las condiciones del entorno.

Más tarde en 1922, el fisiólogo norteamericano Walter B. Canon propone el término *homeostasia* para refererirse a la cualidad de un ser vivo de mantener una relación normal con su medio interno, aun cuando las condiciones de su entorno varíen.

Posteriormente, en 1936 en la Universidad de Montreal el Dr. Hans Selye, a partir de la experimentación con animales, define al estrés como *una respuesta biológica inespecífica del organismo ante cualquier exigencia*. A partir de su tesis, el estrés o síndrome general de adaptación (SGA) se acuña como vocablo para referirse a un conjunto de síntomas psicofisiológicos.

El estudio de sus efectos ha llevado a fisiólogos y psicólogos ha desarrollar mejores estrategias que se emplean para afrontar el estrés, buscando prevenir o controlar los excesos en los eventos procedentes de nuestro entorno, o bien de nosotros mismos. Los métodos psicológicos en el ser humano son considerados como no invasivos, es decir, se realizan mediante cuestionarios o *tests* en los que evalúan diferentes aspectos psicosociales del individuo. Mientras que los métodos fisiológicos analizan el problema creando nuevos dispositivos de medición para observar los cambios en los tejidos y

órganos asociados.

Actualmente se ha desarrollado un nuevo método, que busca determinar el nivel de estrés de una persona. Este tipo de análisis toma en cuenta algunas variables fisiológicas de manera no invasiva y a la vez aplica determinados de *tests* psicológicos. Uno de los precursores de este tipo de análisis en los niveles de estrés fue el Dr. Rispoli [5] quien aplicó pruebas psicológicas y pruebas fisiólogicas en las mismas personas.

En lo que respecta al desarrollo de pruebas fisiológicas, se han realizado estudios para un mejor análisis de las variables involucradas en el estrés. Algunos de éstos se han enfocado en la investigación de las variaciones del ritmo cardiaco, las cuales tienen una estrecha relación con el estado físico de una persona, desarrollándose así sistemas con la capacidad de adquirir e interpretar las variaciones.

1.2 Planteamiento del problema

Este trabajo está basado en el sistema desarrollado por Juan Pablo Flores Ortega en su tesis *"Interfaz avanzada de tiempo real para la medición multidimensional del estrés (IATR-MMS)"*, el cual permitía la medición fisiológica y psicológica del nivel de estrés de un paciente [1].

El interés por el efecto del estrés en la personas llevó al Dr. Adriano de Luca a proponer la creación de un sistema que permitiera medirlo, conjuntado los métodos de análisis utilizados por los médicos y psicológos [4]. Esta construcción condujó a la implementación de un sistema denominado *Sistema Computacional para la Medición Multidimensional del Estrés (SCMME)*[3]; dicho sistema se fue concretando en una serie de investigaciones que fueron añadiendo nuevos elementos para mejorar su desempeño, del cual se resultó el sistema *IATRMMS*.

Este sistema obtiene una medida fisiológica por medio de la técnica de la *fotopletismografía* [9], que permite medir cambios de volumen en consecuencia de variaciones del flujo sanguíneo. Un esquema general del sistema se puede ver en la figura 1.1, en donde se muestran los bloques que conforman el sistema.

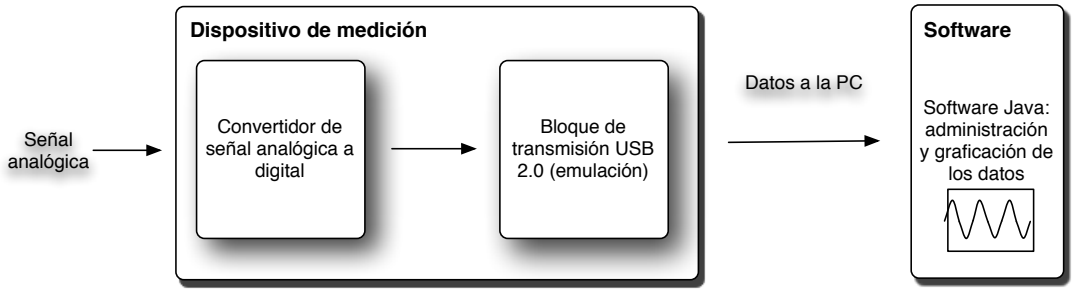

Figura 1.1: Esquema general

En donde la señal analógica es convertida a digital y es enviada a la PC, a través del bloque de transmisión USB. Posteriormente dichos datos son almacenados por el software y graficados en tiempo real. Sin embargo, el sistema tiene deficiencias que deben ser mejoradas y se deben añadir nuevos bloques.

El punto central de esta tesis es la de mejorar en el sistema actual, añadiendo nuevas funcionalidades para mejorar su eficiencia.

Entre algunas de las mejoras que se pretenden llevar acabo, es la de lograr que el módulo de transmisión sea completamente basado en el protocolo de comunicación USB 2.0, debido que el sistema actual solo realiza una emulación de la transmisión por medio de un puerto USB, sin embargo, el software establece la transmisión como si fuera un puerto serie. Lo anterior implica modificar la programación del hardware utilizado para el envío de los datos a la PC, así como el bloque que los recibe de lado del software. Una vez terminado esto, se desarrollará un módulo que permita interpretar los datos recibidos mediante un procedimiento de análisis y finalmente pueda visualizar el resultado del análisis en una gráfica. De ello podemos determinar un esquema global de los propósitos principales de este proyecto, como se ilustra en la figura 1.2.

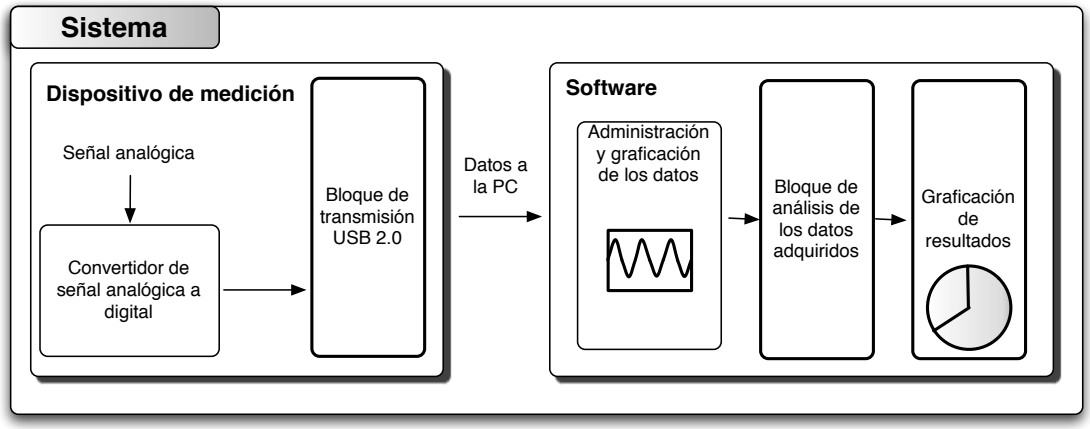

Figura 1.2: Esquema global de los resultados esperados

A pesar que actualmente se tiene un sistema funcional de no ser actualizado, dadas a sus deficiencias, en un futuro podría ser poco funcional o en el peor de los casos obsoleto. Su deficiencia principal recae en la emulación del puerto usb por un puerto serie. Este tipo de puertos, tanto en los nuevos sistemas operativos como en el diseño del hardware en las computadoras de la actualidad, ha dejado de tener soporte y por lo cual paulatinamente ha desaparecido.

Mientras que la transmisión es una parte importante de nuestro proyecto, también lo es el desarrollo de un procedimiento que permita definir el nivel de estrés. Este

procedimiento estará basado en investigaciones sobre el estudio de las variaciones en el ritmo cardiaco y el cual se explicará en los capítulos posteriores de este documento.

1.3 Motivación del proyecto

El sistema de medición propuesto por el Dr. Adriano De Luca, inició con un primer diseño desarrollado por Gregorio Pérez Olán en el 2004 [3]. A partir de entonces, se han realizado cambios radicales al sistema e introducido nuevos elementos que han contribuido en su mejor desempeño.

Para entender mejor la motivación de realizar este proyecto, se mencionarán algunas características e inconvenientes de los diseños anteriores.

Primer diseño:

- Desarrollado por Gregorio Pérez Olán [3].
- Utiliza el estándar USB 1.0 como protocolo de comunicación con la PC.
- El *firmware* fue programado en lenguaje ensamblador.
- Se utilizó un circuito acelerómetro para medir la señal fisiológica abdominal.
- El software fue programado en C++.

Deficiencias del primer diseño:

- Existe pérdida de datos en la comunicación.
- El muestreo de un dato es cada 10ms.
- El diseño no es modular.
- Utiliza costosas tarjetas de adquisición de señales fisiológicas.
- El sistema es invasivo.

Segundo diseño:

- Desarrollado por Enrique Bonilla Henríquez [2].
- Utiliza el estándar USB 1.0 como protocolo de comunicación con la PC.
- El *firmware* fue programado en lenguaje ensamblador.
- El software fue programado en Java.
- El diseño del software fue separado en módulos.
- Se mejoró la velocidad en la transferencia de los datos.

Deficiencias del segundo diseño:

- Se sigue teniendo la pérdida de datos.
- Existe la restricción de la velocidad (22ms) característica del protocolo USB 1.0,
- Utiliza costosas tarjetas de adquisición de señales fisiológicas.
- El sistema sigue siendo invasivo, es decir, el dispositivo de medición es molesto y poco amigable para el usuario.

Tercer diseño:

- Desarrollado por Juan Pablo Flores Ortega [1].
- Utiliza el estándar USB 2.0 como protocolo de comunicación.
- El *firmware* fue programado en Microchip C18 versión 2.42, que es el equivalente de lenguaje C para el microcontrolador.
- El software fue programado en Java.
- Se sustituye las tarjetas por un sensor de PPG (*photoplethysmography* - fotopletismografía)

Deficiencias del tercer diseño:

- El muestreo de la señal no es el estándar usado actualmente.
- Se simula la interfaz USB por medio de un puerto serie, de manera que el protocolo de comunicación no es totalmente USB 2.0.
- El sistema no presenta un análisis de los datos obtenidos.
- La programación en java para el modulo de transmisión lo limita en cuanto a la velocidad de comunicación y transmisión de los datos.

Las deficiencias del último diseño y la necesidad de una actualización al sistema, son la motivación principal para implementar un nuevo bloque de transmisión y otro para la interpretación de la información.

1.4 Objetivos

Analizando las deficiencias heredadas del sistema anterior, así como sus aciertos se decidió desarrollar una nueva versión del sistema.

Para el desarrollo de este nuevo sistema se plantearon los siguientes objetivos generales y específicos, de manera que se cubrieran las deficiencias de la versión anterior.

1.4.1 Objetivos generales

El objetivo general de este proyecto, puede ser visto como dos metas principales.

- Migrar completamente el módulo de transmisión al estándar USB 2.0.
- Desarrollar un bloque de análisis de los datos y otro de graficación de resultados.

1.4.2 Objetivos particulares

Se enlistan los objetivos particulares que servirán como base para lograr cumplir con los objetivos generales. Estos son:

- Utilizar el protocolo USB 2.0 sin la necesidad de emularlo.
- Realizar el diseño e implementación del *firmware* para el funcionamiento con el protocolo USB 2.0.
- Desarrollar el controlador para el dispositivo de medición, cumpliendo con la portabilidad en las diferentes versiones Windows.
- Incrementar la cantidad de muestras de la señal obtenida.
- Desarrollar el bloque de análisis de los datos adquiridos.
- Implementar el bloque que permita la graficación de los resultados de la medición.

1.5 Metodología

El presente proyecto se ha dividido en varias etapas. De manera general, el trabajo de tesis fue planificado para realizarse de manera secuencial. Lo anterior se debe a la cierta dependencia de las etapas finales con las iniciales.

1. Realizar un análisis del sistema anterior utilizado para el control del dispositivo. Enfoncándonos en los bloques de transmisión y administración de datos. Se realizarán pruebas de medición con el dispositivo determinando los alcances del sistema.
2. De manera paralela, realizar un estudio sobre el protocolo de transmisión USB 2.0, con la finalidad de entender su funcionamiento y aplicarlo en el proyecto.
3. Al terminó de los dos primeros pasos, se debe migrar el sistema completamente al protocolo USB 2.0, eliminando la emulación del puerto serie. Esto incluye la modificación del sistema digital encargado de la adquisición de las señal analógica.

4. Para el caso de la transmisión por USB, disminuir el tiempo de conexión entre el dispositivo y software. Aumentar la velocidad de transferencia de los datos.

5. Implementar el módulo de análisis de datos y graficación de resultados, realizando las respectivas pruebas al sistema.

6. Una vez completados los pasos anteriores, se realizarán pruebas de mediciones a personas, para corroborar el correcto funcionamiento del sistema.

1.6 Contenido de los capítulos

El contenido de la tesis está dividido en seis capítulos:

- En el capítulo 1, se presenta una introducción al tema, antecedentes y descripción del problema que dieron origen al estudio de este trabajo. Se describen los objetivos y la metodología para el termino del proyecto.

- En el capítulo 2, se presenta una breve descripción del sensor PPG y el hardware del dispositivo de medición.

- En el capítulo 3, se establecen las bases del protocolo de transmisión USB 2.0, posteriormente la estructura del programa del dispositivo de medición, su funcionamiento y su intervención en la transmisión de los datos. Se describen las herramientas y archivos necesarios para la transmisión..

- El capítulo 4, se presenta el diseño e implementación del sistema, enfoncándonos en los módulos de análisis de los datos y graficación de los resultados.

- El capítulo 5 hace enfásis en las diferentes pruebas realizadas al sistema, resaltando su funcionalidad y los resultados obtenidos.

- En el capítulo 6, se exponen las conclusiones generales y posibles trabajos futuros, que permitan una aportación o innovación más para el sistema digital de medición.

Capítulo 2

Fotopletismografía y hardware del dispositivo de medición

En este capítulo se describe el método utilizado por el dispositivo de medición para obtener la señal analógica y el hardware para la transmisión de los datos a la PC; esto con la finalidad de comprender mejor el funcionamiento general del sistema. Debemos señalar que la descripción de estos tópicos es de manera general, debido a que fueron abordados a detalle en la tesis *Interfaz avanzada de tiempo real para la medición multidimensional del estrés* [1] y que no se realizaron cambios en el hardware.

2.1 Pletismografía

La *pletismografía* incluye técnicas de diagnóstico consistentes en determinar cambios de volumen como consecuencia de variaciones con el flujo sanguíneo. No son métodos específicos de un vaso sanguíneo, arteria o vena, sino que miden cambios de volumen en un segmento de la extremidad [28].

Algunas de estas técnicas se describen a continuación:

- **Pneumopletismografía** o **Pletismografía de volumen de pulso**: Consiste en la colocación de bandas a niveles específicos de la extremidad o en los dedos. Éstos se inflan con una cantidad específica de aire hasta alcanzar una presión entre 10 y 65 mmHg (milímetro de mercurio)[1]. Durante la sístole arterial se produce un incremento en el volumen de la extremidad que transmite presión contra la banda llena de aire, y a través de un sistema transductor de presión, ésta se convierte en una señal analógica de presión.

- **Fotopletismografía**: Detecta el flujo de sangre cutáneo y traduce sus pulsaciones. Consiste en la emisión de luz infrarroja desde un diodo emisor y un fotodetector adyacente que recibe la luz infrarroja reflejada. A medida que aumenta el flujo

[1] mmHg: es la unidad para medir la presión arterial

cutáneo de sangre aumenta la cantidad de luz reflejada. De esta manera obtenemos una medida cualitativa del flujo sanguíneo.

- **Pletismografía por Anillos de Mercurio**: Esta técnica consiste en la colocación de tubos siliconados rellenos de mercurio alrededor de la extremidad. La longitud de estos tubos con mercurio son de 1 a 3 cm, menor que la circunferencia de la extremidad, en caso de tratarse de los dedos esta longitud es menor a 0.5 cm. Su principio físico se basa en las expansiones y contracciones de la extremidad que varían la longitud del tubo de mercurio. Debido a las modificaciones en la resistencia asociados a estos cambios en la longitud, se producen cambios de voltaje ligados a las variaciones de volumen en la circunferencia de la extremidad.

De las anteriores técnicas, el dispositivo de medición utiliza la *fotopletismografía* porque los métodos ópticos son más eficientes, puesto que la profundidad de penetración de la radiacón óptica es pequeña entre 0.1 y 3 milímetros dependiendo de la longitud de onda de la radiación. Esto implica que el análisis de la reflexión de radiación óptica sobre la piel proporciona información selectiva sobre el flujo de la sangre en las capas superiores de la piel, cortando la influencia de las arteries y venas más profundas.

2.2 Fotopletismografía (photoplethysmography)

El principio fiśico de la fotopletismografía está basado en determinar las propiedades ópticas de una área determinada de piel [10]. Para esto, se emite luz infrarroja sobre la piel, la cual es absorbida en mayor o menor cantidad dependiendo de la cantidad del flujo sanguíneo. La luz emitida es reflejada y ésta corresponde con la variación del volumen de sangre. Lo anterior lo podemos visualizar en la figura 2.1.

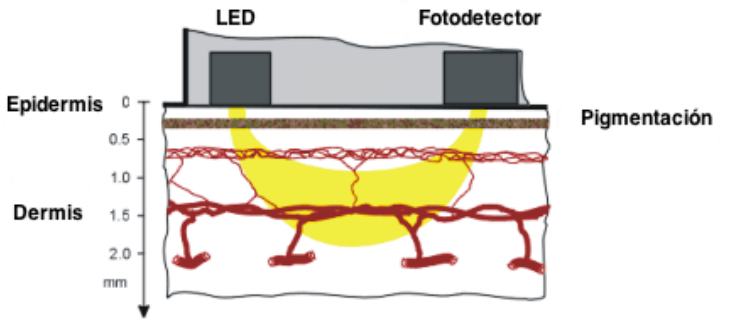

Figura 2.1: Principio físico de la fotopletismografía

En la figura 2.1 se visualiza el uso de sensores para la captura de las variaciones en el volumen del flujo sanguíneo. Existen técnicas que se utilizan con el mismo propósito, variando la posición o materiales de los sensores.

En la siguiente sección se muestra las diversas técnicas y la implementada en el dispositivo de medición.

2.3 Técnicas de captura

El dispositivo obtiene la señal de las variaciones aplicando la fotopletismografía en los dedos, con este próposito se describen los tres tipos de captura [11].

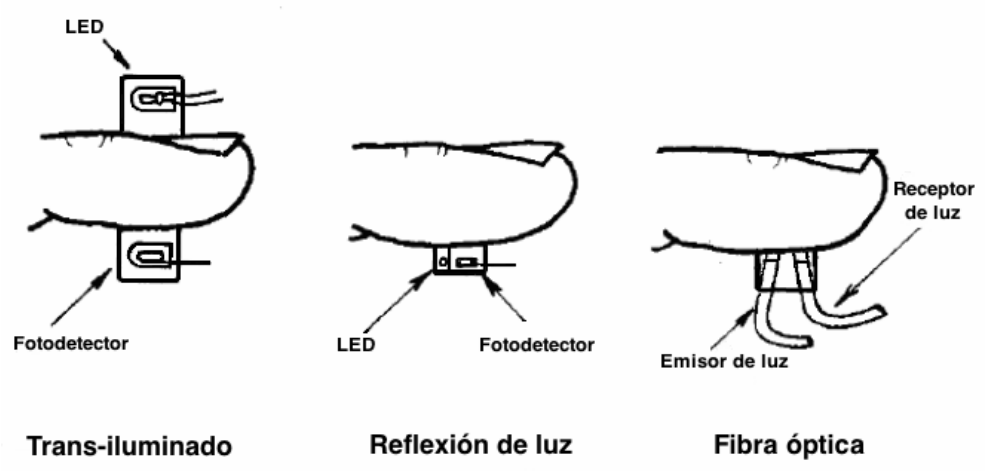

Figura 2.2: Técnicas de captura en fotopletismografía

- **Reflexión de luz**: Esta técnica de captura de señales superficiales detecta la luz reflejada en mayor o menor cantidad por las venas superficiales.
- **Fibra óptica**: Esta técnica utiliza dos cables de fibra óptica en los cuales se transmite un haz de luz infrarroja por uno de los mismos, para que en el otro se detecte por medio de reflexión las variaciones en la señal infrarroja original.
- **Trans-iluminado**: Consiste en colocar los sensores (emisor y detector), de tal forma que el dedo del paciente quede entre los mismos. Es indispensable que la luz infrarroja corresponda al espectro no visible para que la luz reflejada pueda pasar a través del dedo.

La técnica **trans-iluminado** fue la utilizada para implementar la captura de la señal fisiológica, debido a las ventajas que presentaba en el tamaño reducido del emisor como del receptor, teniendo un bajo costo de los mismos.

2.4 Señal característica de la foto-pletismografía

La señal obtenida por cualquiera de las técnicas mencionadas se ilustra en la figura 2.3.

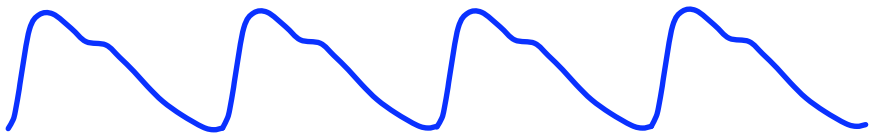

Figura 2.3: Señal característica de la foto-pletismografía

Esta es la señal característica de la foto-pletismografía, la cual está estrechamente relacionada con la *frecuencia cardíaca*. Donde cada elevación en la señal corresponde a una pulsación del corazón. Por ello, la foto-pletismografía es una técnica ampliamente usada para realizar diversos estudios con relación a la frecuencia cardíaca. Entre estos estudios, está el análisis de las variaciones entre las frecuencias cardíacas para medir el nivel de estrés en una persona. Sin embargo, este tema se abordará en el capítulo 4, el cual corresponde al desarrollo del módulo de análisis y graficación.

Por otro lado, el hardware utilizado para la captura de la señal se describe en la siguiente sección.

2.5 Implementación y hardware de la captura de la señal fisiológica

La implementación de la captura de la señal, consta de tres bloques que se muestran en la figura 2.4, sin embargo no fue parte de este trabajo realizarlos, solo se presentan para comprender mejor el funcionamiento del dispositivo de medición. De manera breve, se explica el tratamiento de la señal antes de llegar al convertidor analógico/digital (A/D).

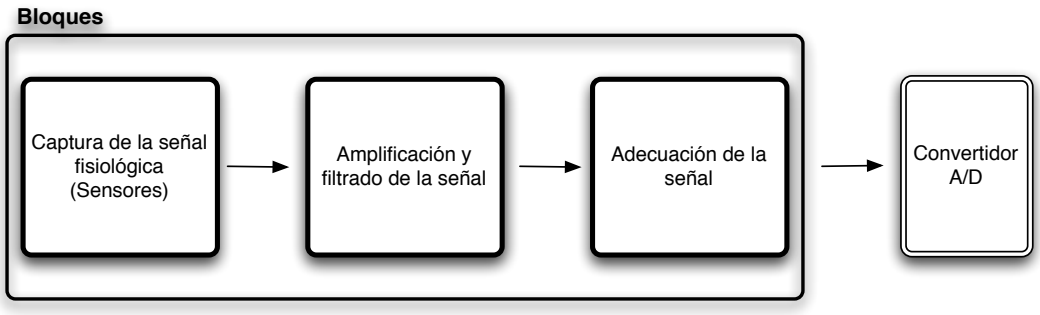

Figura 2.4: Diagrama de bloques: captura de la señal

El bloque de captura de la señal hace uso de un emisor de luz infrarroja TLMK3202 y un diodo fotodetector BPW 34, para generar la señal analógica que es transmitida al

dispositivo de medición. Inmediatamente la señal es amplificada y filtrada eliminando el ruido. Finalmente se traslada la señal que se encuentra oscilando entre valores positivos y negativos, a un nivel positivo en 0 y 5 volts que recibe el convertidor A/D.

2.6 Hardware del dispositivo de medición

Posteriormente a la adquisición, la señal digitalizada es transmitida hacia la PC. Para realizar esta operación, fue necesario utilizar un microcontrolador (MC) capaz de recibir la señal analógica, convertirla a digital y transmitir los datos mediante el uso del protocolo USB 2.0.

Como es sabido existen actualmente varias maneras de implementar el protocolo USB 2.0 a nivel hardware, entre estas se encuentran varios microcontroladores los cuales difieren entre tamaño y características de programación. El bloque de transmisión utiliza el MC 18F4550 de la 18FXXXX de Microchip.

2.6.1 Microcontrolador

El MC 18F4550 fue elegido debido que en trabajos anteriores se utilizó esta familia de MC y principalmente por su compatibilidad con el protocolo USB 2.0. A continuación se enlistan algunas de sus características [29]:

- Memoria de programa Flash.
- Tecnología nanoWatt (consumo muy bajo de corriente).
- Cumple las especificaciones de protocolo USB 2.0.
- Soporta transferencias por control, interrupción, bulk y asíncronas.
- Permite una velocidad total *Full-speed* (12 Mb/s) y una velocidad baja *Low-speed* (1.5 Mb/s).
- 1 Kbyte de memoria RAM dual para la cache de datos USB.
- Módulo transmisor-receptor USB 2.0 independiente.
- Regulador de voltaje USB.
- Resistencias *pull-up* para el módulo USB.

Además de estas características, el MC cuenta con un convertidor A/D de 10 bits con 13 canales, suficiente para digitalizar la señal analógica.

2.6.2 Oscilador y frecuencia de operación

Todo MC requiere de un circuito que le indique a qué velocidad debe trabajar. Este circuito es conocido como un oscilador de frecuencia. En este caso, el MC debe utilizar la frecuencia de 48 Mhz para operar a máxima velocidad. Por lo tanto, el dispositivo dispone de un cristal de 20 Mhz como oscilador externo que pasa por un divisor de 5, teniendo una salida de 4 Mhz que se deriva al reloj interno del módulo PPL de 96 Mhz. Este último tiene una frecuencia de salida de 96 Mhz, la cual pasa por un divisor de 2, obteniendo una frecuencia de reloj interna de 48 Mhz, equivalente a 12 mil instrucciones por segundo (MIPS). El bloque que conforma al oscilador, se muestra en la figura 2.5.

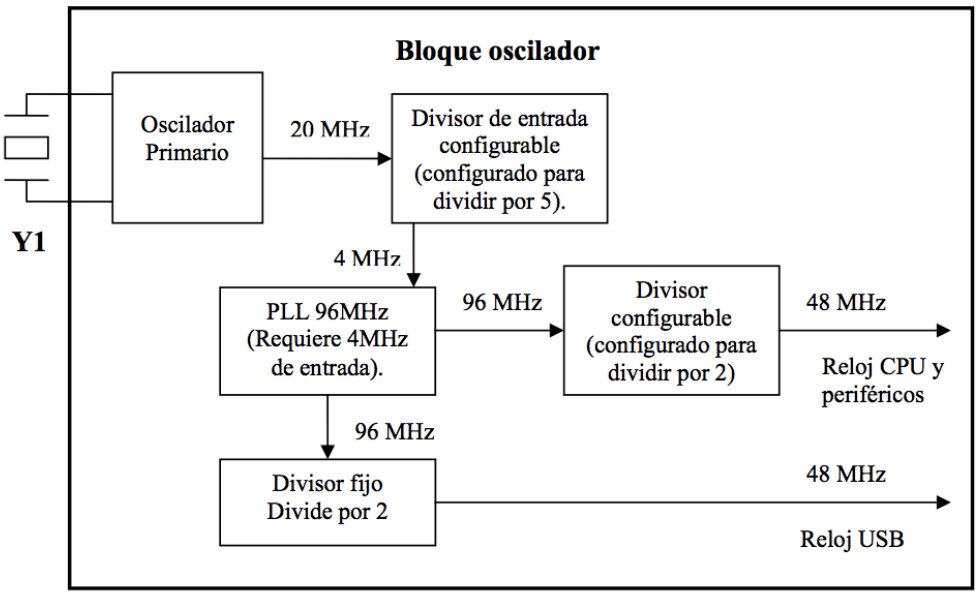

Figura 2.5: Bloque oscilador

2.6.3 Alimentación del MC

El MC funciona con un voltaje de 5V. Este voltaje se obtiene directamente del bus USB. Con esto el dispositivo tiene la ventaja que se puede desacoplar de la alimentación externa del circuito, evitándose la retroalimentación de corrientes entre los circuitos externos y la PC, lo cual elimina el riesgo de choques eléctricos al paciente.

En este capítulo se hizo mención a la principio físico utilizado para obtener la señal fisiológica, así como el hardware necesario y principalmente el hardware con el que se compone el dispositivo de medición. En el siguiente capítulo, se detalla todo el software necesario para llevar acabo la transmisión a través del protocolo USB 2.0.

Capítulo 3

Módulo de transmisión USB 2.0

En este capítulo se detalla el software desarrollado para el módulo de transmisión de datos, a través del protocolo 2.0. Inicia con una breve introducción del protocolo USB, las herramientas necesarias para llevar a cabo la implementación de la transmisión USB, enfocándose principalmente en la explicación de la programación del *firmware* del dispositivo de medición, su controlador y parte del software destinado a la comunicación y recepción de los datos.

3.1 Protocolo USB

El USB o *Universal Serial Bus* es una interfaz para la transmisión serie de datos y distribución de energía desarrollado por empresas líderes del sector de las telecomunicaciones y de las computadoras, ha sido introducida en el mercado de las PC's y periféricos para mejorar las lentas interfaces serie (RS-232) y paralelo. Provee velocidad de transferencia de hasta 100 veces más rápida, comparada con el puerto paralelo de 25-pin y el serial DB-9, DB-25, RS-232. El objetivo original es conectar periféricos relativamente lentos (ratones, impresoras, cámaras digitales, etc.) de una forma sencilla, rápida y basada en comunicaciones serie, aunque por sus características también podían conectarse discos duros.

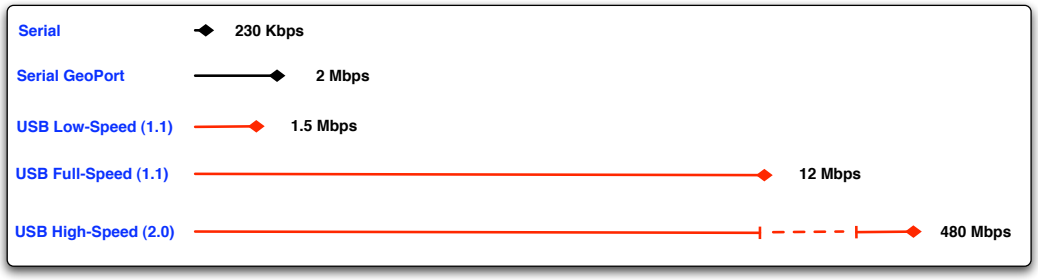

Figura 3.1: Velocidad de transferencia del protocolo USB

Esta interfaz de 4 hilos distribuye 5 volts para la alimentación y puede transmitir datos a una velocidad de hasta 480 Mbps en su versión 2.0, esto equivale a 60 MBytes/s, es decir, la versión 2.0 es 40 veces más rápido respecto a la versión 1.1 (ver figura 3.1). Su bus serie hace posible la conexión de hasta 127 periféricos a un único puerto USB de una PC con detección y configuración automáticas; siendo esto posible con la PC conectada a la red y sin tener que instalar software adicional o reiniciar el sistema (plug and play, algo que no sucedía con los puertos convencionales serie y paralelo) [30, 31, 32].

Las características del protocolo, lo hicieron la opción más viable y eficiente para la transmisión de los datos en el sistema actual y en el sistema propuesto en este documento. Además de permitir la eliminación de la emulación del puerto USB ya mencionada en el capítulo anterior.

Para más detalles respecto a las características del protocolo USB, en el apéndice B se incluye una introducción a sus características y especificaciones técnicas.

Debemos señalar que, debido a la naturaleza del proyecto y a los estándares actuales, se utilizó el protocolo USB en su versión 2.0 y su tipo de transferencia *Bulk* (ver apéndice B.1.4).

El tipo de transferencia *Bulk* fue elegido por estar diseñado para dispositivos que precisan enviar y recibir grandes cantidades de datos, lo cual era conveniente para la transmisión entre el software y el dispositivo de medición.

En las siguientes secciones se presenta el diseño, desarrollo y herramientas necesarias para la implementación del módulo de transmisión USB.

3.2 Diseño del módulo de transmisión USB

Para llevar a cabo la implementación del módulo de transmisión, fue necesario crear un diseño del mismo, que cumpliera con las expectativas y objetivos planteados en el capítulo 1. Con este propósito se planteó un diseño de 5 bloques que conforman el módulo en su totalidad. Se determinó que lo más eficiente era realizar una combinación de los lenguajes de programación Java y C para la implementación del módulo. Cabe remarcar que la transmisión de datos anteriormente estaba implementada en el lenguaje Java en un 100%.

Para nuestro proyecto se intentó eliminar la dependencia de Java en la transmisión, haciendo uso del lenguaje C como herramienta principal para realizar tal actividad, además que el lenguaje C facilita mucho más su programación y es relativamente más rápido que Java.

La realización de una aplicación que combina funciones y métodos de dos lenguajes de programación, conllevó a la necesidad de buscar las herramientas apropiadas para lograr tal fin. De entre las posibles soluciones, se escogieron la API *WinUSB* como la encargada de crear la comunicación y transmisión de datos con el programa empotrado en el microcontrolador (FirmWare) haciendo uso del lenguaje C; y a su vez,

la *Java Native Interface (JNI)* como la herramienta que permite crear funciones nativas en el lenguaje Java con la capacidad de hacer referencias a funciones generadas en el lenguaje C. Ambas herramientas fueron indispensables para la finalización del módulo, primordialmente por su característica de interactuar eficientemente entre ambos lenguajes; y aún más, por facilitar la programación e implementación del protocolo USB 2.0. Para un mejor entendimiento de la implementación del protocolo USB, las siguientes subsecciones se enfocan en las características de las herramientas ya mencionadas.

3.2.1 WinUSB

Proveedores de hardware independientes, que fabrican dispositivos USB, a menudo deben proporcionar una opción para que las aplicaciones puedan acceder a las características del dispositivo. Para el caso de las plataformas Windows, actualmente el desarrollo de aplicaciones con acceso a dispositivos, el *Windows Driver Foundation (WDF)* es el modelo más conveniente para la creación de controladores USB. El *WDF* proporciona tres opciones para facilitar el acceso a un dispositivo USB:

- La implementación de un controlador en modo usuario utilizando el *WDF user-mode-driver (UMDF)* para Windows XP o superior.
- La implementación de un controlador en modo núcleo utilizando *WDF kernel-mode- driver framework (KMDF)* para Windows 2000 o superior.
- Instalación de *WinUsb.sys* como el controlador del dispositivo, proporcionando una aplicación con capacidad de accesar al dispositivo mediante la API de WinUSB para Windows XP o superior .

WinUSB es un controlador genérico USB proporcionado por Microsoft, para sus sistemas operativos a partir de Windows Vista, que también funciona para Windows XP. Antes de Windows Vista, todos los dispositivos USB funcionaban en modo núcleo. Es decir, si se creaba un dispositivo USB y el sistema operativo no tenía el controlador de manera nativa, había que crear un controlador en modo núcleo para el dispositivo [33].

Este controlador genérico está dirigido a dispositivos sencillos que son accesados por una única aplicación a la vez (por ejemplo, instrumentos como las estaciones meteorológicas, los dispositivos que sólo necesitan una conexión de diagnóstico o para actualizaciones de firmware). Permite a la aplicación acceder directamente al dispositivo a través de una simple biblioteca. La biblioteca ofrece acceso a las tuberías del dispositivo. Para ello WinUSB tiene una API que permite a los desarrolladores trabajar con dispositivos USB en modo usuario.

WinUSB consta de dos componentes primarios:

- **WinUsb.sys** es un controlador modo núcleo que puede ser instalado como cualquier controlador de filtro o controlador de función.

- **WinUsb.dll** es una biblioteca de enlace dińamico (*DLL - Dinamic Link Library*) modo usuario, la cual usa la API de WinUSB para crear las funciones de enlace dinámico. Las aplicaciones pueden utilizar esta API para comunicarse con *WinUsb.sys* cuando es instalado como controlador de función de un dispositivo.

Usando la Api WinUSB para comunicarse con un dispositivo es mucho más simple que implementar un controlador, pero tiene las siguientes limitaciones:

- La API WinUSB solo permite que una aplicación pueda comunicarse con el dispositivo al mismo tiempo. Si se desea que más de una aplicación se comunique con él simultáneamente se debe implementar un controlador de función.
- La API WinUSB no soporta las tranferencias isócronas. Este tipo de transferencias requieren de un controlador de función en modo núcleo.
- La API WinUSB no es compatible con dispositivos que cuentan con el soporte en modo núcleo. Ejemplos de estos son los módems y adaptadores de red, que son soportados por la API de telefonía (TAPI) y NDIS, respectivamente.
- Para dispositivos multifuncionales, se puede utilizar el INF del dispositivo para especificar el controlador en modo núcleo o *WinUsb.sys* para cada función USB por separado. Sin embargo, se puede especificar sólo una de estas opciones para cada función en particular, no ambas.

WinUsb.sys también es una parte clave de la relación entre un controlador de función UMDF y el dispositivo asociado. *WinUsb.sys* está instalado en la pila de dispositivos modo núcleo como un controlador de filtro superior. Una aplicación se comunica con el controlador de función UMDF del dispositivo para leer, escribir, o para el control de solicitudes del dispositivo E/S. El controlador pasa la solicitud a *WinUsb.sys*, que procesa la solicitud y la pasa a los controladores del protocolo y finalmente a el dispositivo. Cualquier respuesta se devuelve por el camino inverso. *WinUsb.sys* también sirve como dispositivo de la pila de *Plug and Play* [34, 35].

Especificaciones

WinUSB está soportado por los sistemas operativos:

- Windows Vista en todas sus versiones.
- Windows XP SP2 (Service Pack) y versiones posteriores.

Sin embargo, WinUSB no es nativo de Windows XP, por lo que se tiene que instalar, para los detalles de su instalación refiérase al apartado del apéndice C.2.

Por otro lado, las características USB que soporta WinUSB se enlistan en la tabla 3.1.

Característica	Windows XP	Windows Vista
Control de solicitudes de dispositivos E/S	Soportado	Soportado
Transferencias isócronas	No soportado	No soportado
Tranferencias Bulk, Control y Interrupción	Soportado	Soportado
Suspensión Selectiva	Soportado	Soportado
Encendido remoto	Soportado	Soportado

Tabla 3.1: Tabla de características USB soportadas por WinUSB

En cuanto a la arquitectura de WinUSB en el apéndice C.1 se hace mención a dicho tema; donde se presentan las maneras posibles en que *Winusb.sys* y *Winusb.dll* funcionan ante diferentes dispositivos USB.

Esquema del controlador

En base a las especificaciones de WinUSB, se diseñó un esquema para representar las partes principales que componen al controlador, ver figura 3.2.

Aplicación: Este bloque contiene la aplicación encargada de comunicarse con el dispositivo de medición, utilizando la API de WinUsb. Se desarrolló usando el lenguaje de programación C.

Controlador: El bloque del controlador, está compuesto por los archivos INF y *Winusb.sys*, que en conjunto forman el controlador de función para reconocimiento del dispositivo de medición.

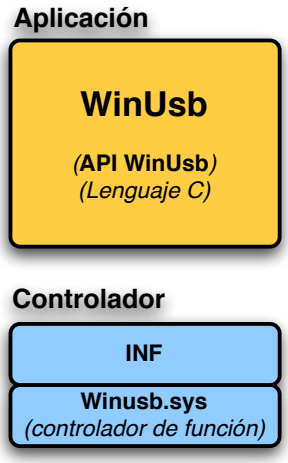

Figura 3.2: Esquema

3.2.2 Java Native Interface (JNI)

La *Java Native Interface* (JNI) es una herramienta de la plataforma Java. Las aplicaciones que usan la JNI pueden incorporar código escrito en lenguajes de programación como C y C++, así como el código escrito en el lenguaje de programación Java.

La plataforma Java y el Host Environment

La plataforma Java es un entorno de programación que se compone por:

- La máquina virtual de Java (JVM)

- Las bibliotecas de Java (API): Consiste de un conjunto de clases predefinidas.

El término *host environment* representa a la computadora que contiene el SO, un conjunto de bibliotecas nativas y el conjunto de instrucciones del CPU. Las aplicaciones desarrolladas en lenguajes de programación como C y C++, compiladas en código binario en un entorno específico y vinculadas a bibliotecas de la JNI, son denominadas como "aplicaciones nativas". Las aplicaciones nativas comúnmente dependen de un *host enviroment*, es decir, su ejecución solo es posible en un SO específico. En caso contrario, la plataforma Java comúnmente está por encima de un *host enviroment*. Por ejemplo, la *Java Runtime Enviroment (JRE)* es un producto de *Sun* que soporta Java en sistemas operativos como Solaris y Windows. Por lo tanto, las aplicaciones desarrolladas en la plataforma java pueden ejecutarse independientemente del *host environment* [13]. Es en este punto, que la JNI tiene un rol importante, para el desarrollo de aplicaciones, que conjuntan la plataforma Java con otros lenguajes de programación.

Papel de la JNI

La JNI permite tener las ventajas de la plataforma Java, pero utilizando código escrito en otros lenguajes. Como una parte de la implementación de la JVM, la JNI tiene una interfaz bidireccional que permite a las aplicaciones Java llamar a código nativo y viceversa. La figura 3.3 ilustra esta interfaz bidireccional [37].

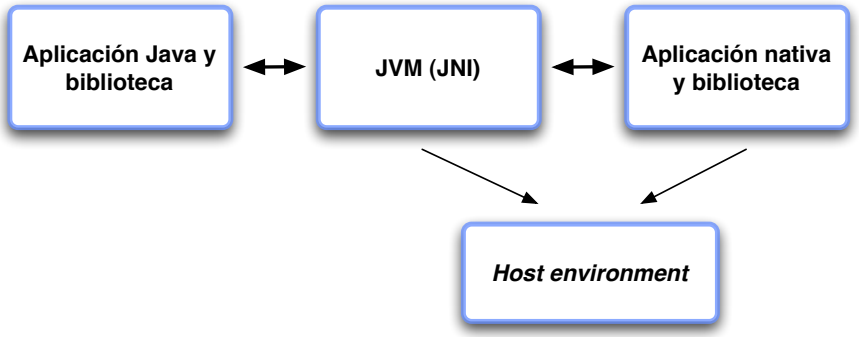

Figura 3.3: Interfaz bidireccional entre Java y las bibliotecas nativas

La JNI está diseñada para manejar las situaciones donde se necesite combinar aplicaciones Java con código nativo. Como una interfaz bidireccional, la JNI puede soportar dos tipos de código nativo: bibliotecas nativas y aplicaciones nativas:

- Con la JNI se puede escribir *métodos nativos* que permiten a las aplicaciones Java llamar funciones implementadas en bibliotecas nativas. Las aplicaciones Java llaman a los *métodos nativos* de la misma manera que llaman a los métodos implementados en Java. Sin embargo, los *métodos nativos* están implementados en otro lenguaje y residen en bibliotecas nativas.

- JNI soporta una interfaz que permite incluir en una aplicación nativa una implementación de la JVM. Las aplicaciones nativas pueden vincularse con una biblioteca que implementa la JVM, y entonces usar el llamado de la interfaz para ejecutar métodos Java en la máquina virtual.

Desventajas de la JNI

Al utilizar JNI en una aplicación se pierden dos beneficios de la plataforma Java. La primera implica que la JNI no pueda ejecutarse en múltiples *host environments*. Aunque la parte de una aplicación escrita en el lenguaje Java es portable a múltiples *hosts*, es necesario recompilar la parte de la aplicación escrita en el lenguaje nativo. La segunda desventaja se muestra cuando el lenguaje Java deja de ser fuertemente tipado, debido que los lenguajes nativos como C o C++ no lo son. Como resultado, se debe tener cuidado al programar aplicaciones que utilicen la JNI. Como regla general, el diseño de la aplicación se debe realizar de manera que los métodos nativos estén definidos en un grupo pequeño de clases. Esto supone un aislamiento entre el código nativo y el resto de la aplicación.

Bibliotecas de enlace dinámico

Cuando se crean aplicaciones con métodos nativos es necesario vincularlas con estos utilizando *bibliotecas de enlace dinámico (DLL)*. Las **bibliotecas de enlace dinámico** son archivos cuyos métodos no se incrustran en el archivo ejecutable durante la fase de enlazado, por el contrario, en tiempo de ejecución la aplicación busca el método, carga su contenido en memoria y enlaza su contenido según va siendo necesario, es decir según se vayan llamando a los métodos [38]. Esto tiene la ventaja de que varias aplicaciones puedan compartir las mismas bibliotecas, lo cual reduce el consumo del disco duro, especialmente con las llamada al SO que suelen ser usadas por muchas aplicaciones a la vez. La extensión de estas bibliotecas varían dependiendo del SO en que se implemente la aplicación. En el caso de Windows tiene como extensión `.dll`.

Descripción general

Para ilustrar el funcionamiento de la JNI, se presenta en la figura 3.4 un ejemplo de una aplicación Java que llama a una funciíon escrita en C para imprimir "`Hola Mundo`". El proceso consiste de los siguientes pasos:

1. Crear una clase (`HolaMundo.java`) que declara el método nativo.

2. Utilizar `javac` para compilar el archivo fuente `HolaMundo`, resultando el archivo de la clase `HolaMundo.class`.

3. Utilizar `javah -jni` para generar un archivo de cabecera en C (`HolaMundo.h`), este contiene prototipo de la función para la implementación del método nativo.

4. Implementar el método nativo en el lenguaje C (`HolaMundo.c`).
5. Compilar la implementación en C del método nativo junto con la biblioteca nativa, creando `HolaMundo.dll`.
6. Ejecutar la aplicación `HolaMundo` usando el intérprete de ejecución `java`. Ambos, la clase (`HolaMundo.class`) y la biblioteca nativa (`HolaMundo.dll`) son cargados en tiempo de ejecución.

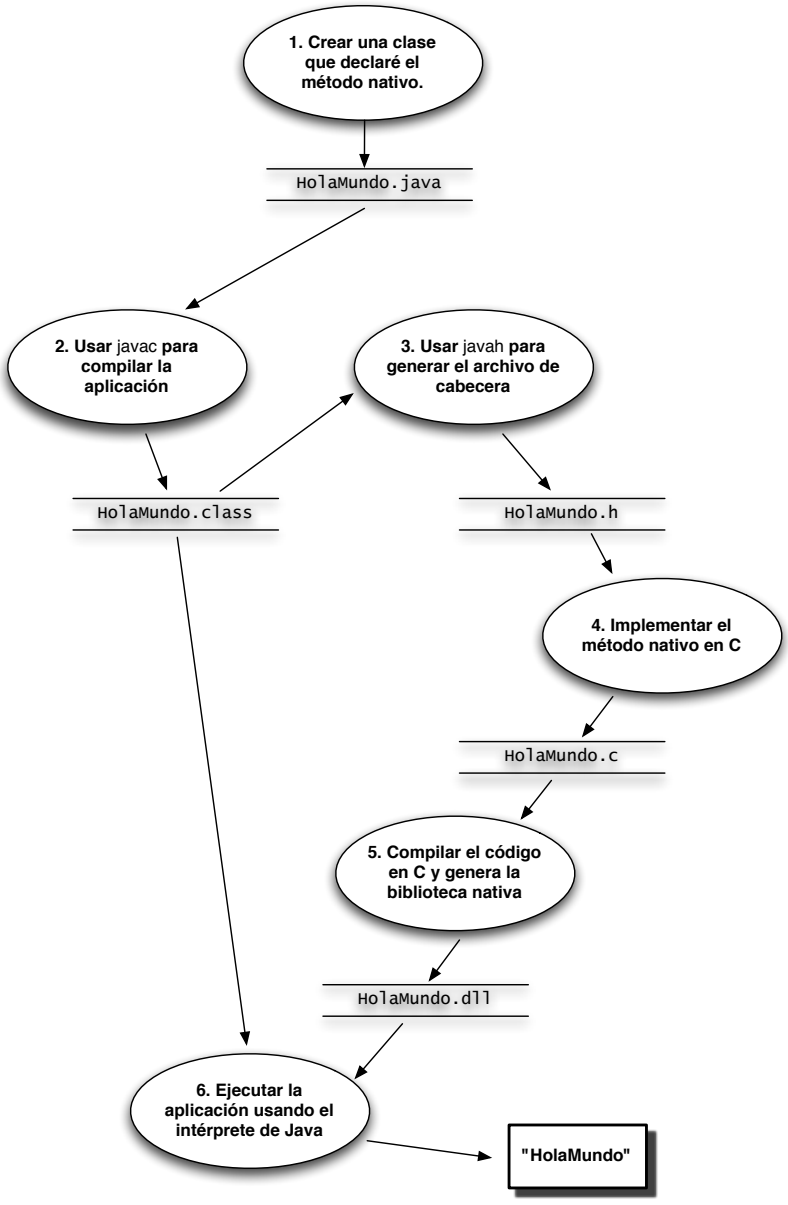

Figura 3.4: Pasos en la escritura y ejecución de la aplicación `HolaMundo`

3.2.3 Esquema final del módulo de transmisión USB

Acorde a las características de las herramientas descritas en las secciones anteriores, se implementó el diseño ilustrado en la figura 3.5, donde se muestran los bloques que conforman al módulo de transmisión. Cada bloque tiene un rol importante para llevar a cabo la tarea de transferir los datos. Cabe señalar que en el esquema se muestran flechas bidireccionales, esto significa que el software tiene el mando del dispositivo al enviar paquetes de control para activar o desactivar el envío de los datos adquiridos desde la señal analógica.

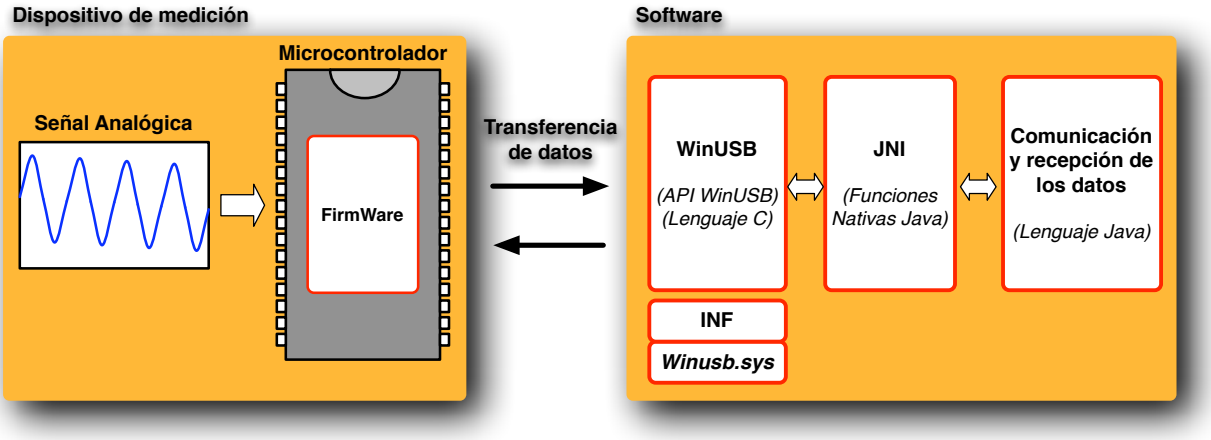

Figura 3.5: Esquema final del módulo de transmisión USB

3.3 FirmWare

El *FirmWare (FW)* es el programa empotrado en el microcontrolador de un dispositivo, este programa se considera parte del hardware, por estar integrado en la electrónica del dispositivo, pero también es software, pues proporciona la lógica y está programado por algún tipo de lenguaje de programación. El *firmware* recibe órdenes externas y responde operando el dispositivo [39].

Para nuestra aplicación el *firmware* se encuentra ubicado en la memoria ROM del microcontrolador PIC18F4450 y se encarga de controlar la transmisión de datos por medio del protocolo USB.

3.3.1 Estructura del FW

La estructura del FW es el conjunto de archivos con los cuales fue construido el FW. Entre estos se encuentran los códigos fuentes y el ejecutable generado para la progra-

mación del MC.

Para el desarrollo de este FW se utilizaron las siguientes herramientas:

- CCS S Compiler, que es el compilador y *debugger* del programa.
- CCS S, es el lenguaje equivalente a C para MC [40].
- Microchip Programador Universal, que es la base para montar el MC y poder descargar el FW.
- Microchip ICD2, que es el programador de MC encargado de verificar si la programación del FW en el MC es correcta.

El proyecto creado para el FW está conformado por varios archivos, la mayoría son generados por el compilador, así que solo se enlistan los más importantes.

- **picwinusb.hex**: es el archivo ejecutable que es utilizado para programar al MC.
- **picwinusb.cof**: este archivo funciona para cargar el programa en simuladores, antes de cargarlo al MC.
- **picwinusb.c**: es el archivo fuente encargado del control de la activación de dispositivo, así como la transferencia de los datos al *host*.
- **picwinusb.h**: es el archivo fuente que contiene la configuración del dispositivo USB.

Estos cuatros archivos conforman la base del proyecto, en especial los dos últimos que contienen el código fuente y son la base de la estructura lógica del FW.

3.3.2 El FW en la estructura lógica

La estructura lógica del FW lo conforman los archivos *picwinusb.c* y *picwinusb.h*, los cuales se encargan de la configuración del descriptor USB y del envío de los paquetes de datos. Al utilizar el lenguaje CCS S como un equivalente al lenguaje C facilita la programación de los microcontroladores, es por ello que el FW se reduce a la programación de estos dos archivos.

A continuación estos archivos serán discutidos a fin de saber que hace cada uno de ellos.

3.3.3 `picwinusb.c`

Este archivo es el encargado de configurar el *endpoint*, la inicialización del dispositivo y la activación o desactivación del envío de paquetes dependiendo de las acciones del *host*.

El código fuente es el siguiente y posteriormente se exponen las funciones de cada directiva del código.

```
#include <18F4550.h>
#include <pic18_usb.h>
#include <PicWinUSB.h>
#include <usb.c>
//-------------------------------------------------------------
#fuses HSPLL,NOWDT,NOPROTECT,NOLVP,NODEBUG,USBDIV,PLL5,
CPUDIV1, VREGEN
//-------------------------------------------------------------
#use delay(clock=48000000)
//-------------------------------------------------------------
#define USB_HID_DEVICE     FALSE
#define USB_EP1_TX_ENABLE  USB_ENABLE_BULK
#define USB_EP1_RX_ENABLE  USB_ENABLE_BULK
#define USB_EP1_TX_SIZE    2
#define USB_EP1_RX_SIZE    2
//-------------------------------------------------------------

void main(void) {

   int8 iBuff[2];
   int8 oBuff[2];

   setup_adc_ports(AN0);
   setup_adc(ADC_CLOCK_INTERNAL);
   set_adc_channel(0);

   usb_init();
   usb_task();
   usb_wait_for_enumeration();
//-------------------------------------------------------------

   while (TRUE)
   {
      if(usb_enumerated())
      {
         if (usb_kbhit(1))
          {
            usb_get_packet(1, iBuff, 2);

               do{
                  oBuff[0] = 1;
                  oBuff[1] = read_adc();
```

```
                delay_us(3850);
                usb_put_packet(1, oBuff, 2, USB_DTS_TOGGLE);
              }while(usb_kbhit(1)!=TRUE);
          }
       } // if
    } // while
} // main
//-------------------------------------------------------------
```

Includes

Picwinusb.c incorpora los siguientes archivos de cabecera necesarios para su correcta compilación:

#include <18F4550.h>: Permite incluir la biblioteca donde se encuentran definidos identificadores de los registros del modelo de MC a programar.

#include <pic18_usb.h> : Capa hardware de los dispositivos PIC18Fxx5x. Incluye la biblioteca que contiene las funciones y constantes para la comunicación USB con los MC de la familia 18Fxx5x de Microchip para el controlador USB de CCS.

#include <PicWinUSB.h>: Incluye el archivo que contiene los descriptores del dispositivo USB.

#include <usb.c> : Manejador de peticiones USB estándar. Es otra biblioteca con más funciones generales sobre la comunicación USB, maneja las interrupciones.

Fuses y frecuencia del MC

Un *fuse* es un fusible. Hay ciertos aspectos del PIC que han de ser activados o desactivados mediante hardware a la hora de programarlo. Esto quiere decir que no se pueden volver a cambiar hasta que el chip no se reprograme de nuevo. Para el proyecto, es necesario habilitar algunos de estos *fuses* y cada uno tiene una función [14].

NOWDT: deshabilita el *Watch Dog Timer* (WDT). El WDT se activa al iniciarse el programa del MC y es el encargado reiniciar el MC en un intervalo de tiempo determinado (32ms), en caso que ocurra una falla o problema de software. En esta caso se desabilita por que no es necesario usarlo [41].

HSPLL: habilita al cristal de cuarzo a 20Mhz. Ver figura 2.5 del capítulo 2.

NOPROTECT: habilita el *fuse* que indica que es un código que no está protegido de lectura.

NOLVP: habilita el bajo voltaje de programación.

NODEBUG: deshabilita en modo de depuración para ICD.

USBDIV: indica que la fuente de reloj USB viene de dividir PLL entre 2. Ver figura 2.5 del capítulo 2.

PLL5: divide lo 20Mhz del cristal de cuarzo entre 5, teniendo como resultado 4Mhz utilizados por el PLL. Ver figura 2.5 del capítulo 2.
CPUDIV1: divide los 96MHz del PLL entre 2 = 48MHz reloj sistema. Ver figura 2.5 del capítulo 2.
VREGEN: activa el regulador de voltaje USB.

#use delay(clock= 48000000): permite definir la frecuencia del oscilador del MC. El compilador lo utiliza para realizar cálculos de tiempo. Esta frecuencia es la misma que se mencionó en el capítulo 2, la cual es de 48Mhz.

Configuración de los *endpoints*

Para configurar los *endpoints* es necesario del uso de las directivas `define`. Los *endpoints* se pueden ver como buffers temporales de datos de entrada/salida. Están asociados a las tuberías que conectan al *host*, además que son bidireccionales (ver figura 3.6) [42].

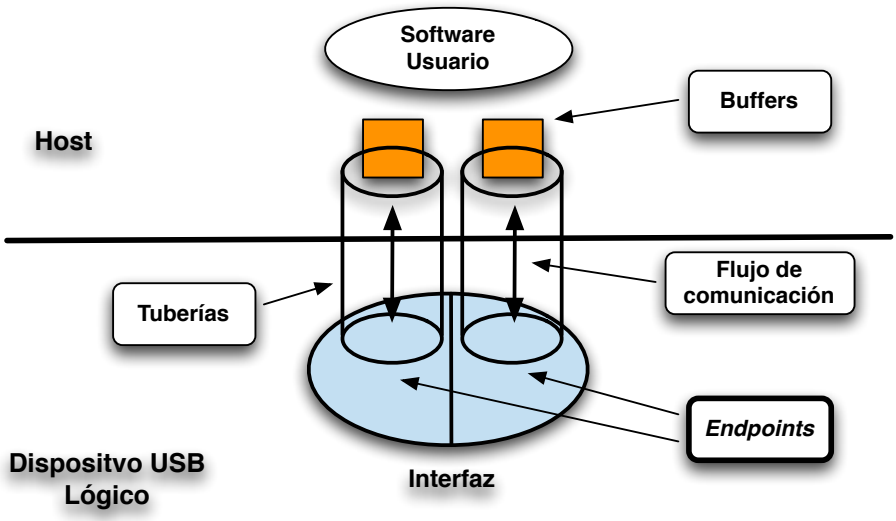

Figura 3.6: Representación de los *endpoints* en el flujo de comunicación USB

```
#define USB_HID_DEVICE     FALSE
#define USB_EP1_TX_ENABLE  USB_ENABLE_BULK
#define USB_EP1_RX_ENABLE  USB_ENABLE_BULK
#define USB_EP1_TX_SIZE    2
#define USB_EP1_RX_SIZE    2
```

#define USB_HID_DEVICE FALSE : Indica que el tipo de dispositivo a programar no es de interfaz humana (HID). Los dispositivos HID son aquellos que son reconocidos por los sistemas operativos sin la necesidad de un controlador especial. Por *default* el compilador CCS trae está opción habilitada para dispositivos HID, por tal motivo es necesario deshabilitarla.

#define USB_EP1_TX_ENABLE USB_ENABLE_BULK y **#define USB_EP1_RX-_ENABLE USB_ENABLE_BULK**: Estas dos instrucciones definen a un *endpoint* EP1 que funcionará con transacciones de tipo bulk (Apéndice B).

#define USB_EP1_TX_SIZE 2 y **#define USB_EP1_RX_SIZE 2**: Definen el tamaño del *endpoint* EP1 en paquetes de 2 bytes. Su tamaño puede ser extendido de acuerdo a la necesidad de la aplicación del *host* con las limitantes que imponga el tipo de transacción.

Declarando variables e inicializando el dispositivo USB

Dentro la función `main()` se declaran las variables necesarias con las que el dispositivo envían los bytes al *host*, se abre el puerto por el cual el MC lee los datos del convertidor A/D y se crea el flujo de comunicación entre el MC y el *host* previamente al inicio de transmisión de los bytes.

```
int8 iBuff[2];
int8 oBuff[2];

setup_adc_ports(AN0);
setup_adc(ADC_CLOCK_INTERNAL);
set_adc_channel(0);

usb_init();
usb_task();
usb_wait_for_enumeration();
```

int8 iBuff[2] y **int8 oBuff[2]**: Declaran los buffers que contendrán los bytes de entrada y salida respectivamente. Tienen un tamaño de 2 bytes, el mismo que tiene declarado el *endpoint*. Su función principal es almacenar los bytes transmitidos por medio de la tubería hacia el *endpoint*.

setup_adc_ports(AN0): Determina que el puerto `AN0` del MC será un puerto analógico. Debemos señalar que en esta familia de MC, los puertos pueden ser de entrada analógica o digital. Por este puerto la señal analógica tendrá acceso al convertidor

A/D.

setup_adc(ADC_CLOCK_INTERNAL): Determina el tiempo de muestreo de la señal analógica en base al reloj interno del MC.

set_adc_channel(0): Indica que el convertidor A/D tomará la lectura en el canal 0. Los MC pueden tener diferentes entradas analógicas, para ello poseen multicanales y por tal motivo es necesario especificar que canal utilizará el convertidor para leer.

usb_init(): Esta función inicializa el dispositivo USB. Espera en un ciclo infinito hasta que el periférico USB es conectado al bus. Habilita y utiliza la interrupción USB.

usb_task(): Inicializa el periférico USB. Si se utiliza una detección de conexión para la inicialización, entonces se debe llamar periódicamente a esta función para controlar el pin de detección de la conexión. Cuando el MC es conectado o desconectado del bus, esta inicializa el periférico USB.

usb_wait_for_enumeration(): Esta función se queda en un ciclo infinito hasta que el *host* habilite al dispositivo. El *host* habilita al dispositivo cuando es enumerado y entonces entra en modo de operación normal y puede enviar y recibir paquetes de datos.

Recepción y envío de paquetes

La recepción y envío de paquetes está a cargo de un ciclo infinito que siempre está checando la recepción de un paquete por parte del *host* que habilite el envío de los paquetes. Al recibir otro paquete por parte de *host*, este indicaría que el envío de paquetes deber ser suspendido, regresando a su estado inicial de espera.

```
while (TRUE) {
     if(usb_enumerated()) {
        if (usb_kbhit(1)) {
            usb_get_packet(1, iBuff, 2);
            do{
               oBuff[0] = 1;
               oBuff[1] = read_adc();
               delay_us(3850);
               usb_put_packet(1, oBuff, 2, USB_DTS_TOGGLE);
            }while(usb_kbhit(1)!=TRUE);
        }
     }
}
```

usb_enumerated(): Esta función devuelve verdadero si el dispositivo ha sido enumerado por el *host*. A diferencia de `usb_wait_for_enumeration()`, esta no se queda en un ciclo infinito.

usb_kbhit(1): La función devuelve verdadero si el *endpoint* de entrada contiene paquetes del *host*. En caso que tuviera paquetes, significa que el *host* a habilitado el inicio de transmisión de paquetes.

usb_get_packet(1, iBuff, 2): Esta función copia el contenido del *endpoint* de entrada en el buffer *iBuff*. El primer argumento de la función es el *endpoint* donde llega el paquete de datos, el segundo es el apuntador donde guardamos ese dato y el tercer argumento es el tamaño de paquetes en bytes.

Las siguientes instrucciones se ubican dentro de un ciclo que tiene como condición de salida, recibir un paquete desde el *host* que deshabilte la lectura de datos del convertidor A/D.

do{...} while(usb_kbhit(1)!=TRUE): Ciclo que se repite mientras no reciba un paquete desde el *host* indicando el alto al envío de paquetes.

read_adc(): Función que realiza la lectura desde el convertidor A/D, devolviendo un valor que puede ser almacenado en una variable. En este caso, el valor devuelto es almacenado en el buffer de salida `oBuff`.

usb_put_packet(1, oBuff, 2, USB_DTS_TOGGLE): Realiza la función de enviar los bytes almacenados en el buffer de salida hacia el *host* por medio del *endpoint*.

3.3.4 `picwinusb.h`

Este archivo es el más importante del FW, debido que contiene la configuración y descriptores del dispositivo USB.

Los descriptores son estructuras de datos o bloques de información con determinado formato que permiten al *host* obtener las características del dispositivo USB. Los descriptores usados en los dispositivos USB pueden clasificarse según las partes o funciones del dispositivo que describen. Cada descriptor contiene un valor constante y único que identifica su tipo [15, 43].

En la tabla 3.2 se muestran los tipos de descriptores utilizados para el FW de nuestro dispositivo, mientras que la figura 3.7 ilustra la estructura en que se encuentran ubicados los descriptores en el archivo `picwinusb.h`.

Tipo de Descriptor	Descripción
Dispositivo	Contiene la información básica del dispositivo
Configuración	Todos los dispositivos tienen al menos una configuración y por lo mismo un descriptor de la misma
Cadena	Este descriptor contiene información sobre el fabricante, el producto u otros aspectos del dispositivo. A cualquier sistema USB se le pueden agregar una gran cantidad de descriptores de cadena, para hacerlo más presentable o explícito
Interfaz	En los sistemas USB, interfaz se refiere a un conjunto de *endpoints* usados para determinada función. Cada configuración debe soportar al menos una interfaz y pueden haber varias interfaces activa al mismo tiempo al seleccionar una configuración. Cada interfaz tiene su descriptor y consecuentemente los *endpoints* de cada interfaz tienes sus descriptores propios
Endpoint	El EP0 no tiene descriptor, pero los demás *endpoints* deben proveer información sobre el tamaño máximo de paquetes además de otros datos

Tabla 3.2: Tabla de tipos de descriptores USB

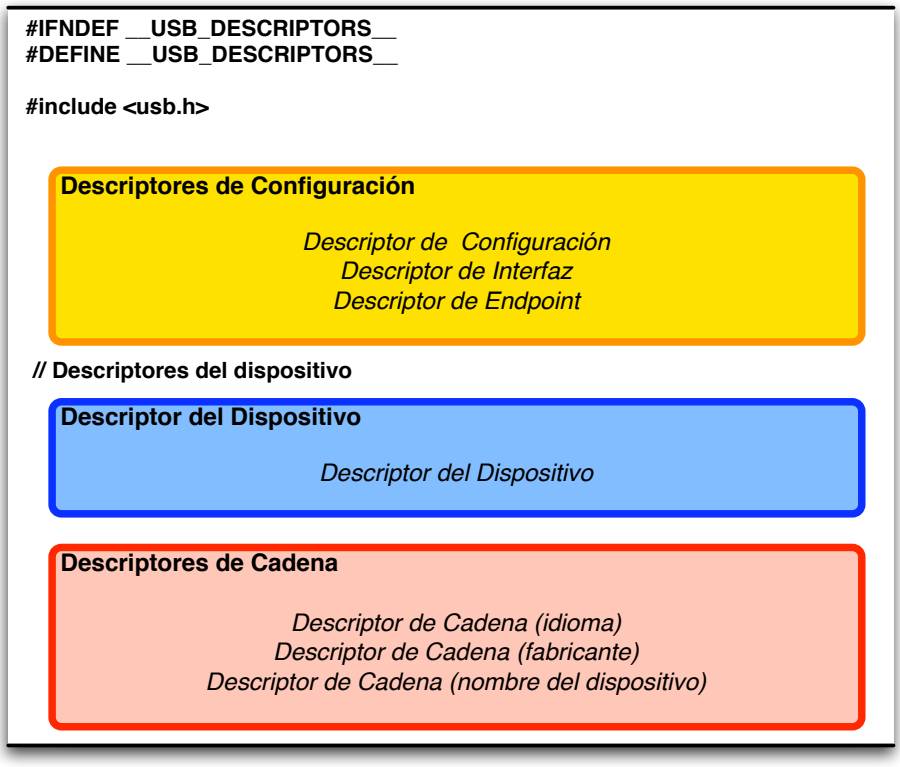

Figura 3.7: Descriptores del dispositivo USB en `picwinusb.h`

Los descriptores son declarados en arreglos de tipo *chars* predefinidos por la biblioteca de funciones USB(`usb.h`). Estos arreglos se componen por uno o varios descriptores de diferentes tipos colocados de manera consecutiva. A continuación se explican los campos de los descriptores en el orden mostrado en la figura 3.7.

Descriptores de Configuración

Se describen 4 descriptores en el arreglo `USB_CONFIG_DESC[]`, estos son de tipo configuración, interfaz y dos de *endpoint* (tabla 3.3).

Campo	Descripción
Descriptor de configuración	
`USB_DESC_CONFIG_LEN`	Tamaño del descriptor (bytes)
`USB_DESC_CONFIG_TYPE`	Constante que indica el tipo de descriptor (0x02h)
`USB_TOTAL_CONFIG_LEN, 0`	Tamaño del total de los datos que serán enviados por la configuración en bytes
`1`	Número de interfaces soportadas
`0x01`	Identificador para las solicitudes de configuración
`0x00`	Índice de cadena para descriptor de configuración
`0xC0`	Atributos de *wakeup* y potencia de bus
`0x32`	Potencia requerida para el bus (0x32 = 100mA)
Descriptor de Interfaz	
`USB_DESC_INTERFACE_LEN`	Tamaño del descriptor (bytes)
`USB_DESC_INTERFACE_TYPE`	Constante de descriptor de interfaz (0x04h)
`0x00`	Número de interfaz
`0x00`	Valor establecido para alternar características
`2`	Número de *endpoints* exceptuando EP0
`0xFF`	Código de clase (FF = definido por vendedor)
`0xFF`	Código de subclase
`0xFF`	Código de protocolo
`0x00`	Índice del descriptor para la interfaz
Descriptor de Endpoint (EP1 TX y EP1 RX)	
`USB_DESC_ENDPOINT_LEN`	Tamaño del descriptor (bytes)
`USB_DESC_ENDPOINT_TYPE`	Constante de descriptor de interfaz (0x05)
`0x81`	Número de *endpoint* y dirección (0x81 = EP1 entrada)
`0x01`	Número de *endpoint* y dirección (0x01 = EP1 salida)
`0x02`	Tipo de transferencia soportado (2 = bulk)
`USB_EP1_TX_SIZE, 0x00`	Máximo tamaño en paquetes para EP1 TX
`USB_EP1_RX_SIZE, 0x00`	Máximo tamaño en paquetes para EP1 RX
`0x01`	Periodos determinados en ms

Tabla 3.3: Tabla de los descriptores de configuración

Descriptor del Dispositivo

Este descriptor está definido dentro del arreglo `USB_DEVICE_DESC[]`. Almacena la información básica del dispositivo (tabla 3.4).

Campo	Descripción
Descriptor del dispositivo	
`USB_DESC_DEVICE_LEN`	Tamaño del descriptor (bytes)
`0x01`	Constante de descriptor de dispositivo
`0x10, 0x01`	Número de versión USB en bcd
`0x00`	Código del protocolo a usar
`USB_MAX_EP0_PACKET_LENGTH`	Máximo tamaño de paquetes para el EP0
`0xD8, 0x04`	ID del fabricante (0x4D8 de Microchip)
`0x11, 0x00`	ID del producto
`0x01`	Índice del descriptor de cadena del fabricante
`0x02`	Índice del descriptor de cadena del dispositivo
`0x00`	Índice del descriptor que contiene el número serial
`USB_NUM_CONFIGURATIONS`	Número de posibles configuraciones

Tabla 3.4: Tabla del descriptor del dispositivo

Descriptores de cadena

Los descriptores de cadena se almacenan en el arreglo `USB_STRING_DESC[]`. En dicho arreglo contiene tres de estos descriptores que definen el idioma, nombre del fabricante y el nombre del dispositivo (tabla 3.5).

Campo	Descripción
Descriptor de cadena (idioma)	
`4`	Tamaño del descriptor (bytes)
`USB_DESC_STRING_TYPE`	Constante de descriptor de cadena (0x03h)
`0x09, 0x04`	Código de idioma (Inglés definido por Microsoft)
Descriptor de cadena (nombre del fabricante)	
`20`	Tamaño del descriptor (bytes)
`USB_DESC_STRING_TYPE`	Constante de descriptor de cadena (0x03h)
`'StressUSB'`	Nombre del fabricante
Descriptor de cadena (nombre del dispositivo)	
`20`	Tamaño del descriptor (bytes)
`USB_DESC_STRING_TYPE`	Constante de descriptor de cadena (0x03h)
`'PicWinUSB'`	Nombre del dispositivo

Tabla 3.5: Tabla de descriptores de cadena

3.4 Controlador del dispositivo (archivo INF)

El archivo INF generado es esencial para la correcta instalación del dispositivo, siendo su función principal contener los valores necesarios para el reconocimiento del dispositivo y asignación al driver *winusb.sys* por parte del SO.

En seguida se muestra el archivo INF de nuestro dispositivo.

```
; ================ Version section =================
[Version]
Signature = "$Windows NT$"
Class = %ClassName%
ClassGuid={16180339-8874-9894-8482-045868343656}
Provider = %ProviderName%
DriverVer = 15/03/2009,1.0.0.0

; ========== Manufacturer/Models sections ===========
[Manufacturer]
%ProviderName% = PicWinUSB,NTamd64

[PicWinUSB]
%DeviceDesc% =WinUSB_Install, USB\VID_04D8&PID_0011

[PicWinUSB.NTamd64]
%DeviceDesc% =WinUSB_Install, USB\VID_04D8&PID_0011

; ================== Installation ==================
;[1]
[WinUSB_Install]
Include=WinUSB.inf
Needs=WinUSB.NT

;[2]
[WinUSB_Install.Services]
Include=WinUSB.inf
AddService=WinUSB,0x00000002,WinUSB_ServiceInstall

;[3]
[WinUSB_ServiceInstall]
DisplayName     = %WinUSB_SvcDesc%
ServiceType     = 1
StartType       = 3
ErrorControl    = 1
ServiceBinary   = %12%\WinUSB.sys

;[4]
[WinUSB_Install.Wdf]
KmdfService=WinUSB, WinUSB_Install

[WinUSB_Install]
KmdfLibraryVersion=1.7

;[5]
[WinUSB_Install.HW]
AddReg=Dev_AddReg

[Dev_AddReg]
HKR,,DeviceInterfaceGUIDs,0x10000,"{31415926-5358-9793-2384-626433832795}"

;[6]
[WinUSB_Install.CoInstallers]
AddReg=CoInstallers_AddReg
CopyFiles=CoInstallers_CopyFiles
```

```
[CoInstallers_AddReg]
HKR,,CoInstallers32,0x00010000,"WdfCoInstaller01007.dll,WdfCoInstaller","WinUSBCoInstaller.dll"

[CoInstallers_CopyFiles]
WinUSBCoInstaller.dll
WdfCoInstaller01007.dll

[DestinationDirs]
CoInstallers_CopyFiles=11

; ============= Source Media Section ===========
;[7]
[SourceDisksNames]
1 = %MediaDescription%

[SourceDisksFiles]
WinUSBCoInstaller.dll=1, x86
WdfCoInstaller01007.dll=1, x86

[SourceDisksFiles.NTamd64]
WinUSBCoInstaller.dll=1, amd64
WdfCoInstaller01007.dll=1, amd64

; ================== Strings ==================
[Strings]
ProviderName="StressUSB"
ClassName="StressUSB"
DeviceDesc="PicWinUSB"
MediaDescription="Dispositivo PicWinUSB"
WinUSB_SvcDesc="Dispositivo PicWinUSB Stress"
```

Las principales partes del archivo INF se describen a continuación conforme la enumeración presentada en el código fuente del archivo. Antes se debe señalar que la construcción del INF es acorde al ejemplo suministrado por Microsoft para el controlador WinUSB [35].

1. Las directivas `Include` y `Needs` en la sección *WinUSB_Install* son necesarios para la instalación de WinUSB en los sistemas Windows Vista. Los sistemas Windows XP ignoran estas directivas.

2. La directiva `Include` en la sección *WinUSB_Install.Services* incluye el archivo INF proporcionado por el sistema para WinUSB. El INF es instalado por *WinUSBCoInstaller* si no está en el sistema destino. La directiva `AddService` específica a *Winusb.sys* como el controlador de función del dispositivo.

3. La sección *WinUSB_ServiceInstall* contiene los datos para la instalación de *Winusb.sys* como un servicio.

4. La directiva `KmdfService` en la sección *WinUSB_Install.Wdf*. La sección de *WinUSB_Install* específica la versión de la biblioteca KMDF. Este archivo INF se basa en la versión de Windows Vista de la WDK, que incluye la versión KMDF 1.7.

5. *WinUSB_Install.HW* es la sección clave en el INF; ésta especifíca el identificador global único de la interfaz del dispositivo (GUID). La directiva `AddReg` coloca la

interfaz GUID en un registro de valor estándar. Cuando *Winusb.sys* es cargado como un controlador de función, se lee el registro y utiliza el GUID especificado para representar la interfaz del dispositivo. Es importante que el GUID sea el mismo para la aplicación del *host*.

6. La sección *WinUSB_Install.CoInstallers*, incluidas las secciones `AddReg` y `CopyFiles`, contienen los datos y las instrucciones para instalar WinUSB e instaladores KMDF y asociarlos con el dispositivo.

7. Se definen los diferentes `co-installers` para las versiones x86 y x64 de Windows.

Podemos destacar una de las partes primordiales del archivo INF es la declaración del GUID, con la cual la aplicación accesa al controlador del dispositivo. Pero también hay una parte igual de importante que las mencionadas, esta se ubica en la siguiente línea de código:

```
USB\VID_04D8&PID_0011
```

En ella se puede observar la definición del VID y PID, en este caso **VID=04D8** y **PID=0011**. Dichos valores son los mismos declarados en el FW del dispositivo, los cuales son los proporcionados por Microchip para el MC, sin ellos el controlador no sabría identificar el tipo de dispositivo a conectarse.

3.5 Implementación de las funciones de transmisión (WinUSB y JNI)

Las funciones encargadas de la comunicación entre el dispositivo USB y el software de la aplicación, son una combinación entre las funciones de la API WinUSB y las funciones nativas creadas para la transmisión. El conjunto de estas funciones componen a la biblioteca de enlace dinámico (*winusb.dll*) necesaria para la transmisión entre la aplicación y el FW del dispositivo.

La *winusb.dll* fue renombrada a *ComPPGDevice.dll* con la finalidad de tener un nombre más acorde a nuestra aplicación. *ComPPGDevice.dll* está conformada por tres archivos fuentes: **ComPPGDevice.cpp**, **ComPPGDevice.h** y **Conexion.h**; como se muestra en el esquema de la figura 3.8.

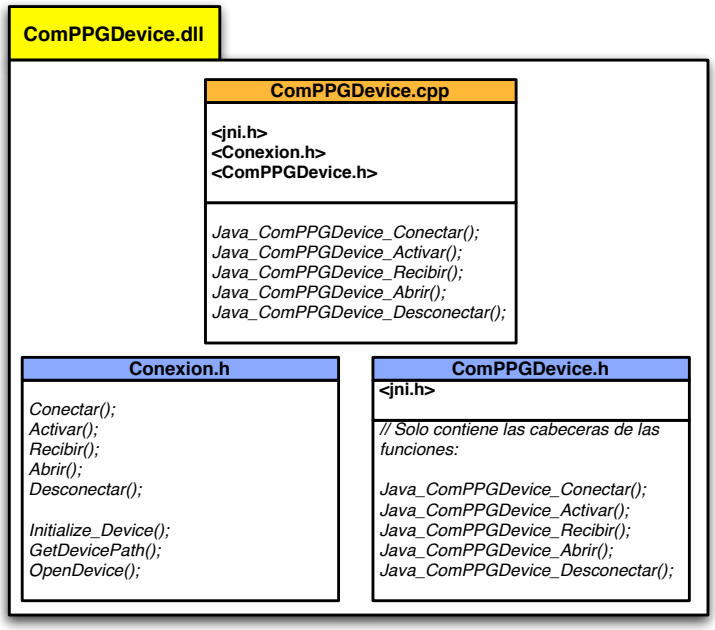

Figura 3.8: Archivos fuentes que conforman a *ComPPGDevice.dll*

3.5.1 ComPPGDevice.cpp

La biblioteca dinámica *ComPPGDevice.dll*, es el archivo principal que contiene el cuerpo de las funciones nativas construidas con la JNI. Estas funciones nativas son el puente entre los lenguajes Java y C, para ello la JNI tiene definidos en la biblioteca `jni.h` tipos de datos equivalentes entre ambos lenguajes. Esto permite manejar los tipos de datos como en el lenguaje C, o bien, soportan la recepción de valores devueltos por funciones escritas concretamente en C. En la tabla 3.6 se muestra esta equivalencia de tipos.

Tipo Java	Tipo C	Descripción
`boolean`	`jboolean`	8 bits sin signo
`byte`	`jbyte`	8 bits con signo
`char`	`jchar`	16 bits sin signo
`shor`	`jshort`	16 bits con signo
`int`	`jint`	32 bits con signo
`long`	`jlong`	64 bits con signo
`float`	`jfloat`	32 bits formato IEEE
`double`	`jdouble`	64 bits formato IEEE

Tabla 3.6: Tipos de datos fundamentales Java y C

La posibilidad de poder invocar desde las funciones nativas a funciones escritas en C, nos permite reducir el código en las primeras. Este es el caso de las funciones descritas en este archivo fuente, la cual comparten una estructura similar aunque cada una con diferente función.

Estas funciones tienen una forma particular de ser declaradas. Sus nombres indican la clase de Java que las invoca, por ejemplo la siguiente declaración:

```
JNIEXPORT jboolean JNICALL Java_ComPPGDevice_conectar (JNIEnv*
env, jobject obj)
```

La directiva `JNIEXPORT jboolean` indica que es una función nativa y devuelve un valor booleano, mientras que `JNICALL Java_ComPPGDevice_conectar` señala que la función es invocada desde la clase `ComPPGDevice` y la función nativa se llama `conectar`.

A continuación describimos las funciones nativas:

- **Java_ComPPGDevice_conectar**: Esta función devuelve un valor del tipo `jboolean`. Invoca a la función `conectar` ubicada en la biblioteca `Conexion.h`, para inicializar el dispositivo.

```
    jboolean resp;
    resp = conectar();
    return resp;
```

- **Java_ComPPGDevice_activar**: Devuelve un valor del tipo `jboolean`. Invoca a la función `activar`, para indicar al dispositivo el inicio de transmisión de los datos.

```
    jboolean resp;
    resp = activar();
    return resp;
```

- **Java_ComPPGDevice_recibir**: Recibe un puntero del tipo `jbyteArray`. Este puntero corresponde a un arreglo del tamaño del buffer descrito en el FW del dispositivo (2 bytes). Posteriormente invoca la función `recibir`, para leer los datos recibidos y actualiza el arreglo.

```
jbyte* datos;
  datos = (jbyte*)env->GetByteArrayElements(buffer,NULL);
  recibir(datos);
  env->ReleaseByteArrayElements(buffer,datos,0);
```

- **Java_ComPPGDevice_abrir**: Retorna un valor del tipo `jboolean`, invocando a la función de `abrir` para verificar si el dispositivo se encuentra transmitiendo datos.

```
jboolean resp;
resp = abrir();
return resp;
```

- **Java_ComPPGDevice_desconectar**: Retorna un valor del tipo `jboolean`, para indicar que el dispositivo ha desactivado la transmisión de datos. Para ello invoca la función `activar`, la cual sirve para activar y desactivar la transmisión.

```
jboolean resp;
resp = activar(); // Activa y desactiva la transmision
return resp;
```

Cabe remarcar que todas las funciones invocadas por las funciones nativas, se ubican en el archivo **Conexion.h** implementadas en el lenguaje C.

3.5.2 ComPPGDevice.h

Este archivo es generado autómaticamente por la JVM al compilarse la clase `ComPPGDevice`. De acuerdo a las especificaciones de la JNI, este archivo generado debe quedar inalterable. Contiene las cabeceras de las funciones nativas contenidas en **ComPPGDevice.cpp** .

3.5.3 Conexion.h

El archivo **Conexion.h** es el más importante de los que conforman la DLL. En él se definen las funciones que permiten el acceso al dispositivo USB utilizando la API WinUSB.

La API ofrece un conjunto de funciones para realizar la conexión con el dispositivo y otro conjunto para el envío y recepción de bytes del mismo.

Funciones de conexión

La API proporciona este conjunto de funciones para inicializar el dispositivo. De ellas son tres las principales que intervienen en este proceso. Pero para tal fin, es de suma importancia definir la GUID del dispositivo, que identifica al dispositivo como único.

La manera definir el GUID está dada de la siguiente manera:

```
DEFINE_GUID(InterfaceClassGuidConstant,0x31415926, 0x5358,
0x9793, 0x23, 0x84, 0x62, 0x64, 0x33, 0x83, 0x27, 0x95);

// Archivo INF:  HKR,,DeviceInterfaceGUIDs,0x00010000,"{31415926
// -5358-9793-2384-626433832795}"
```

El GUID debe ser el mismo al especificado en el archivo INF utilizado para instalar a *Winusb.sys*, de no ser así, le sería imposible a la aplicación comunicarse con el dispositivo.

Por otro lado, las funciones para realizar la conexión son las siguientes:

- **Initialize_Device()**: Inicia el proceso de conexión con el dispositivo. Obtiene valores como la velocidad del dispositivo, el tamaño de los *endpoints*, el tipo de transferencia, etc. Para obtener los valores mencionados, necesita de un archivo identificador (HANDLE) para el dispositivo, el cual obtiene invocando la función `OpenDevice`. Posteriormente inicializa la API con la función `WinUsb_Initialize`.

```
HANDLE deviceHandle = OpenDevice(TRUE);
 bResult = WinUsb_Initialize(deviceHandle, &usbHandle);
```

- **OpenDevice()**: Esta función crea el archivo identificador del dispositivo, donde se definen algunos atributos como el soporte de lectura y escritura síncrona en el dispositivo. La generación de este archivo es solo posible si se obtiene la ruta del dispositivo, para ello invoca a la función `GetDevicePath()`.
- **GetDevicePath()**: Se encarga de encontrar la ruta del dispositivo con la ayuda del GUID, definido anteriormente.

Por último todo el proceso de conexión es iniciado por la función `conectar`, al recibir por parte de la aplicación la inicialización del dispositivo. Dado que la función **Initialize_Device** retorna un valor booleano, se puede determinar si el dispositivo está conectado.

Funciones de envío y recepción de bytes

Este conjunto está conformado por solo dos funciones. Una de ellas es para la envío de bytes a través de la tubería hacia el *endpoint* (descrito en la sección del FW) y la otra para la recepción de bytes provenientes del dispositivo.

- **WinUsb_WritePipe()**: Envía un paquete de bytes a través de la tubería hacia el *endpoint* de entrada. El tamaño del paquete depende del tamaño definido en el FW para el *endpoint*. En este caso el tamaño es de 2 bytes. Retorna un valor booleano para indicar si el envío tuvo éxito o hubo error.

  La función `activar` utiliza esta función de la API para iniciar la transmisión de datos.

- **WinUsb_ReadPipe()**: Recibe un paquete de bytes enviado por el dispositivo. Al igual que *WinUsb_WritePipe()* el paquete recibido debe ser del tamaño definido para el *endpoint* del FW. El paquete recibido lo almacena en un buffer del mismo tamaño para ser enviando a la aplicación.

  Las funciones `recibir` y `abrir` hacen uso de ella, la primera para recibir los bytes de manera continua y la segunda para verificar si el dispositivo está activo.

Todas las funciones de la API, están especificadas en la documentación proporcionada por Microsoft [35].

3.6 Control y recepción de los datos

El bloque encargado para el control y recepción de los datos, está ubicado al inicio de todo el proceso de la transmisión de los datos. Al ejecutarse la aplicación, este bloque se encargar de verificar si el dispositivo está conectado y de controlar la activación o desactivación de la transmisión de los datos.

En este caso, la clase **ComPPGDevice.class** es la encargada de llevar a cabo este control. En ella se declaran las funciones nativas almacenadas en la biblioteca *ComPPGDevice.dll*.

```
private native boolean conectar();
private native boolean activar();
private native void recibir(byte[] readbuffer);
private native boolean abrir();
private native boolean desconectar();
```

Las funciones anteriores son las mismas declaras en la DLL. Sin embargo, para que la aplicación haga uso de ellas, es necesario cargar la biblioteca en memoria. Para ello se realiza una llamada al SO para verificar su tipo y así tener la certeza del uso de la biblioteca apropiada.

Al referirnos sobre la selección de la biblioteca apropiada, es debido que se tuvo la necesidad de generar dos DLL, *ComPPGDeviceXP.dll* y *ComPPGDeviceVista.dll*, para Windows XP y Windows Vista respectivamente. La implementación de bibliotecas separadas para cada SO, tiene como fundamento en las especificaciones de WinUSB, donde se señala esta separación por la diferencia entre las arquitectura de ambas plataformas. El siguiente fragmento de código de la clase, muestra la condicional para seleccionar la DLL correspondiente a cada SO.

```
String os = System.getProperty("os.name");
s = os.toLowerCase();

        if(os.startsWith("windows vista"))
             System.loadLibrary("ComPPGDeviceVista");
        else
            System.loadLibrary("ComPPGDeviceXP");
```

3.6.1 Control de la transmisión de datos

Para tener el control de la conexión del dispositivo, así también como activar o desactivar la transmisión de datos, la clase implementa las siguientes funciones:

- `ComPPGDevice`: Es el constructor de la clase, al generarse un objeto, crea una conexión con el dispositivo y activa la transmisión de datos. Hace uso de las funciones nativas `conectar` y `activar`.

```
    public ComPPGDevice() {
      System.out.println("Crea la conexion...");
      if (conectar()){
        System.out.println("Conexion realizada...");
        if(activar())
          System.out.println("Transmision iniciada...");
        else
          System.out.println("Error al activar dispositivo...");
     }
     else
```

```
        System.out.println("Error en la conexion");
    }
```

- `openDevice`: Esta función verifica si el dispositivo está activo o transmitiendo datos invocando la función nativa `abrir`.

```
    public synchronized void openDevice() {
       if (abrir())
         System.out.println("Se ha abierto el dispositivo...");
       else
         System.out.println("Error al abrir el dispositivo...");

       hiloTrans.start();
    }
```

- `closeDevice`: Se encarga de desactivar la transmisión de datos. Para ello invoca a la función nativa `desconectar`.

```
    public synchronized void closeDevice() {
      if(desconectar())
        System.out.println("Se ha cerrado el dispositivo...");
      else
        System.out.println("Error al cerrar el dispositivo...");
    }
```

3.6.2 Recepción de datos

La recepción de datos está cargo de un hilo de ejecución, el cual constantemente verifica la entrada de datos desde el dispositivo. Su implementación se muestra en el siguiente fragmento de código.

```
public void run() {
    byte readBuffer[] = new byte[2];
    while(true){
        recibir(readBuffer);
```

```
        BigInteger datoRecibido = new BigInteger(1, readBuffer);
        fireNewDataAvailable(datoRecibido.longValue());
    }
}
```

A través de un buffer de tamaño de 2 bytes, recibe los datos de manera continua, posteriormente realiza una conversión de bytes a un entero. Finalmente la función `fireNewDataAvailable` toma el valor del datos recibido y lo envía a una tabla de valores, donde es almacenada para que otro bloque del sistema se encargue de su procesamiento.

Capítulo 4

Módulo de análisis y graficación

Este capítulo describe la implementación del módulo de análisis de datos y graficación de resultados. Presenta una breve descripción de las bases teóricas en que se sustenta el procedimiento que permite realizar un diagnóstico de nivel de estrés. Más adelante, de manera independiente, se detallan las funciones e importancia de cada bloque que conforman al módulo.

4.1 Diseño del módulo de análisis y graficación

El diseño del módulo de análisis y graficación está conformado por tres bloques: **grabación**, **análisis** y **graficación**. Cada bloque tiene su función principal en base al procedimiento desarrollado para medir el estrés de un individuo.

Previo a la descripción de cada uno de los bloques, se exponen las bases teóricas fundamentales para la implementación del proceso de análisis de datos y diagnóstico de resultados.

Estas bases teóricas, son estudios médicos comprobados que fundamentan un posible análisis de variables fisiológicas, para determinar una cierta medida cualitativa de estrés. El estudio concerniente a este proyecto, es el análisis de las variaciones en la frecuencia cardiaca.

4.1.1 Variabilidad de la frecuencia cardíaca

La *variabilidad de la frecuencia cardíaca* son las variaciones de los intervalos de tiempo entre cada pulsación del corazón. Estas variaciones son objeto de estudio para determinar el estado de salud de una persona, principalmente para el estudio del sistema nervioso [44].

Para realizar un análisis de las variaciones, es necesario obtener los tiempos R-R; estos se denominan así debido a la nomenclatura utilizada para definir una señal de electrocardiograma (EGG), donde el pulso del corazón está denotado como QRS, como se muestra en la figura 4.1.

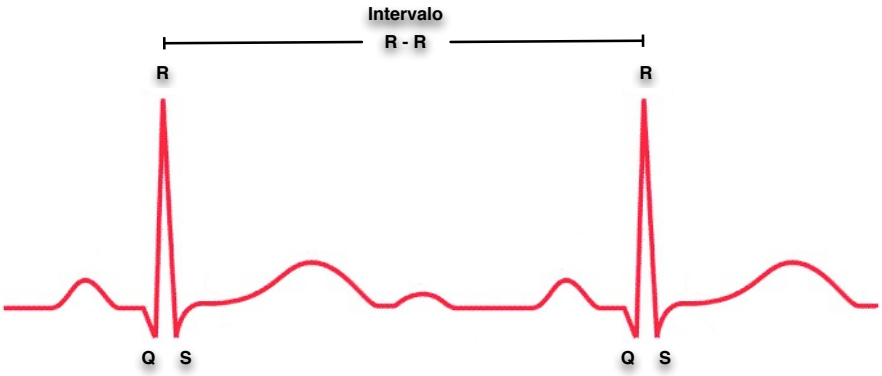

Figura 4.1: Intervalo de tiempo R-R

Con los intervalos R-R es posible obtener las diferentes frecuencias con las que trabaja el corazón, y las cuales están relacionadas con el sistema nervioso [19].

4.1.2 Relación entre las frecuencias y el sistema nervioso

Para entender la relación entre las frecuencias y el sistema nervioso, debemos explicar que éste último está conformado por dos estados, el estado *Simpático* y el *Parasimpático*; ambos son más conocidos como **Sistema Nervioso Simpático (SS)** y **Sistema Nervioso Parasimpático (SP)**.

Dado que este proyecto no se enfoca directamente al estudio del sistema nervioso, no se explica a detalle la función de los SS y SP. Solo es necesario decir que ambos sistemas regulan el comportamiento físico de algunos órganos y funciones del cuerpo humano. Por lo tanto, el desbalance entre estos sistemas provocado por diversos factores, principalmente por el estrés, conducen a problemas físicos o psicológicos en un individuo. En otras palabras, la medición del estrés recae en el análisis del balanceo de los sistemas SS y SP [18, 20].

Por lo anterior, algunas investigaciones han demostrado que la variabilidad de la frecuencia cardiaca es el factor principal para medir los sistemas SS y SP, concluyendo que las bajas frecuencias corresponden a un cierto estado del SS y las altas frecuencias a el SP.

En base a tales investigaciones, se ha podido desarrollar un procedimiento para medir los estados de los sistemas SS y SP, y por lo tanto obtener una medida del nivel de estrés.

4.1.3 Procedimiento de medición

A continuación se describe de manera general el procedimiento desarrollado para realizar una medición de estrés:

1. Adquisición de la señal fisiológica: Para este paso se utiliza la técnica de la *fotopletismografía* utilizando el dispositivo de medición, descrito en el capítulo 3 de este documento.

2. Calculó de los tiempos R-R: Se calculan los tiempos R-R, sin embargo, la señal característica de la *fotopletismografía* no es igual a una señal EGG, por ello fue necesario calcular la derivada de la señal para obtener los valores picos de ella, obtiendo así una equivalencia con los tiempos R-R de una señal EGG. La equivalencia se puede observar en la figura 4.2.

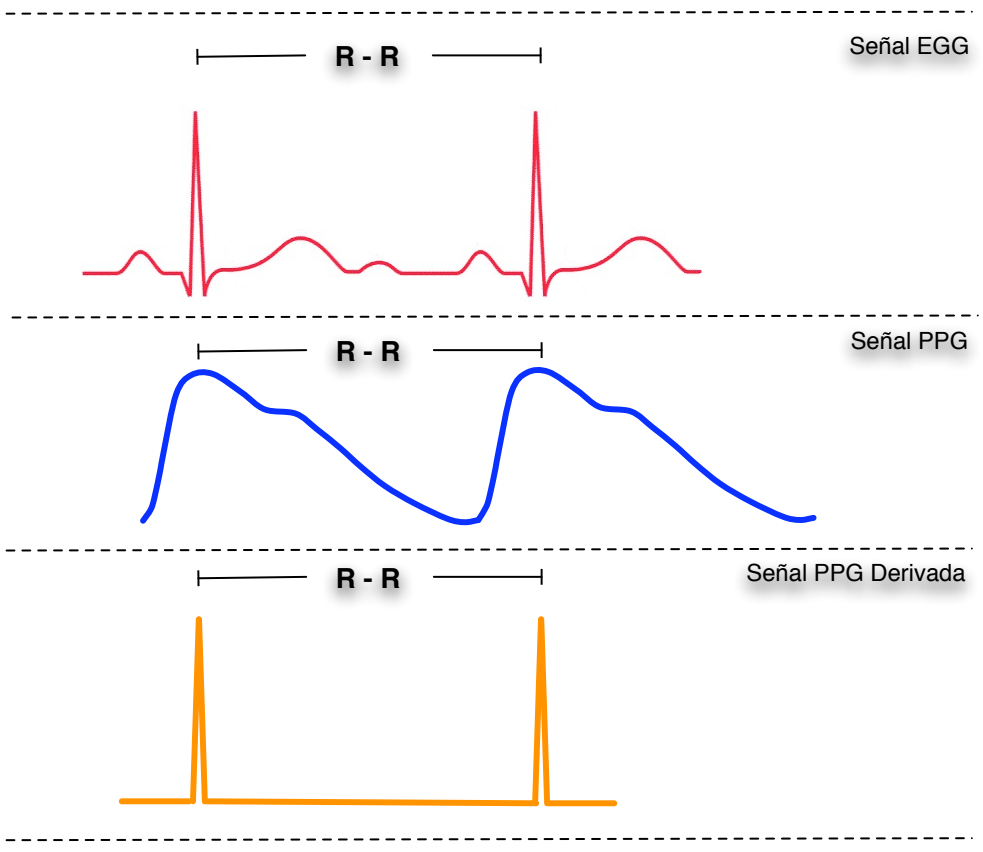

Figura 4.2: Equivalencia entre las señales EGG y PPG

3. Grabación de los tiempos R-R: Se almacenan los tiempos R-R en durante un tiempo determinado.

4. Análisis de los tiempos R-R: Se calcula el espectro de frecuencias en base a los tiempos R-R calculados, con la finalidad de encontrar las bajas y altas frecuencias que determinan el balanceo entre los sistemas SS y SP.

5. Graficación de resultados: De acuerdo a los resultados, se expone en una gráfica el diagnóstico del nivel de estrés.

4.1.4 Esquema del módulo de análisis y graficación

Acorde al procedimiento descrito, se diseño el siguiente esquema que representa al módulo de análisis y graficación (ver figura 4.3). Donde se observan los tres bloques principales mencionados.

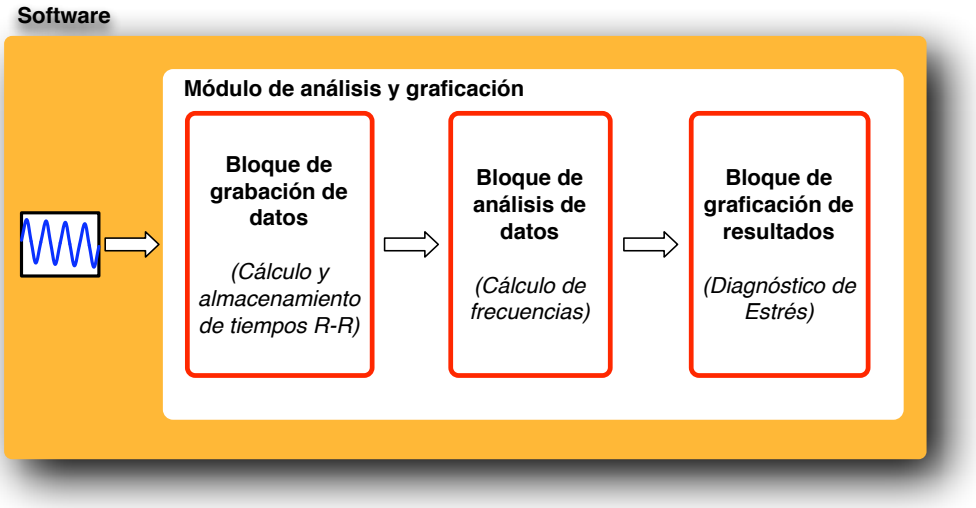

Figura 4.3: Esquema del módulo de análisis y graficación

En las siguientes secciones se describen las implementaciones de cada uno de los bloques, resaltando sus funciones principales y la relación entre ellos.

4.2 Implementación del bloque de grabación de datos

El **bloque de grabación** es el encargado de calcular los tiempos R-R y almacenarlos para su posterior análisis. Para realizar este proceso, se llevan a cabo tres funciones principales. La primera de ellas es calcular la derivada de la señal para obtener los tiempos R-R, la segunda es el almacenamiento de los tiempos y conteo de pulsos cardiacos, y finalmente la graficación de la señal en tiempo real durante la grabación.

Su implementación está dada por las clases que se muestran en la figura 4.4.

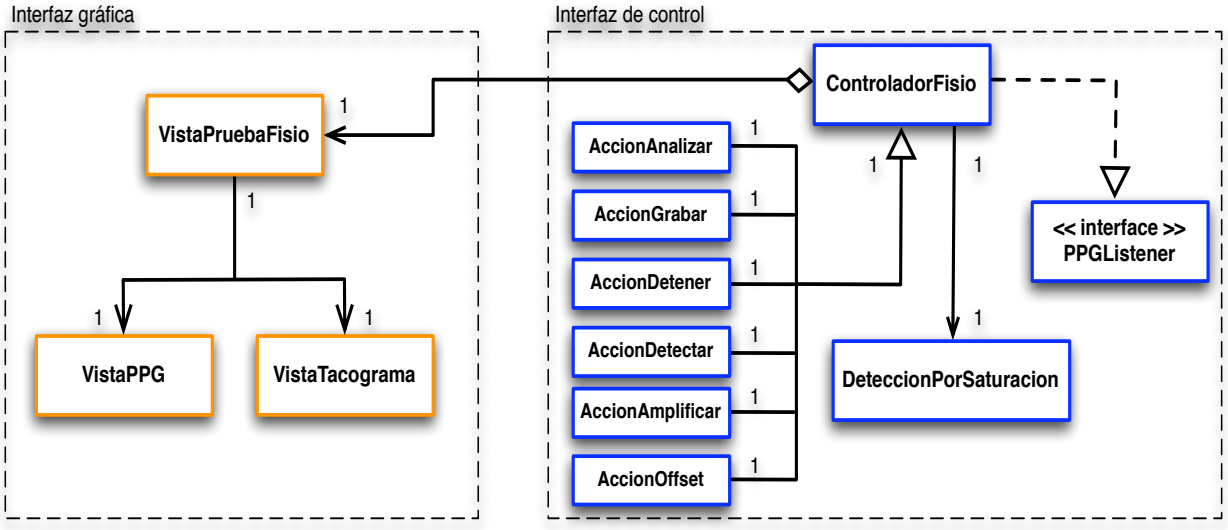

Figura 4.4: Diagrama de clases del bloque de grabación

En el diagrama se desglosan dos grupos de clases, las clases que conforman a la interfaz gráfica y las de la interfaz de control. Las clases de la interfaz gráfica, como su nombre lo indica, se encargan del manejo de los gráficos mostrados durante el proceso de grabación. Las clases de la interfaz de control, son las encargadas de controlar la recepción de la señal, calculando los picos más altos a través de su derivada y por consiguiente obtener los intervalos de tiempos R-R.

4.2.1 Interfaz gráfica del bloque de grabación

La interfaz gráfica está conformada por tres clases, que asumen el papel de contener todos los componentes gráficos y funciones necesarios para la graficación de la señal en tiempo real. El diagrama de clases presentado en la figura 4.5 muestra los principales métodos y objetos con las que están constituidas las tres clases.

VistaPruebaFisio: Esta clase es la principal, en ella se engloban todos los componentes para construir la interfaz gráfica del bloque de grabación. Tiene una importante relación con la clase `ControladorFisio` de la interfaz de control, dado que es la clase por la cual se instancian las funciones de graficación. Incluye los botones de gráficos, que permiten al usuario controlar las acciones de la interfaz previamente, durante y posteriormente a una grabación. De la misma manera, agrega un objeto de las clases `VistaPPG` y `VistaTacograma`, encargadas de la graficación de la señal y del tacograma respectivamente.

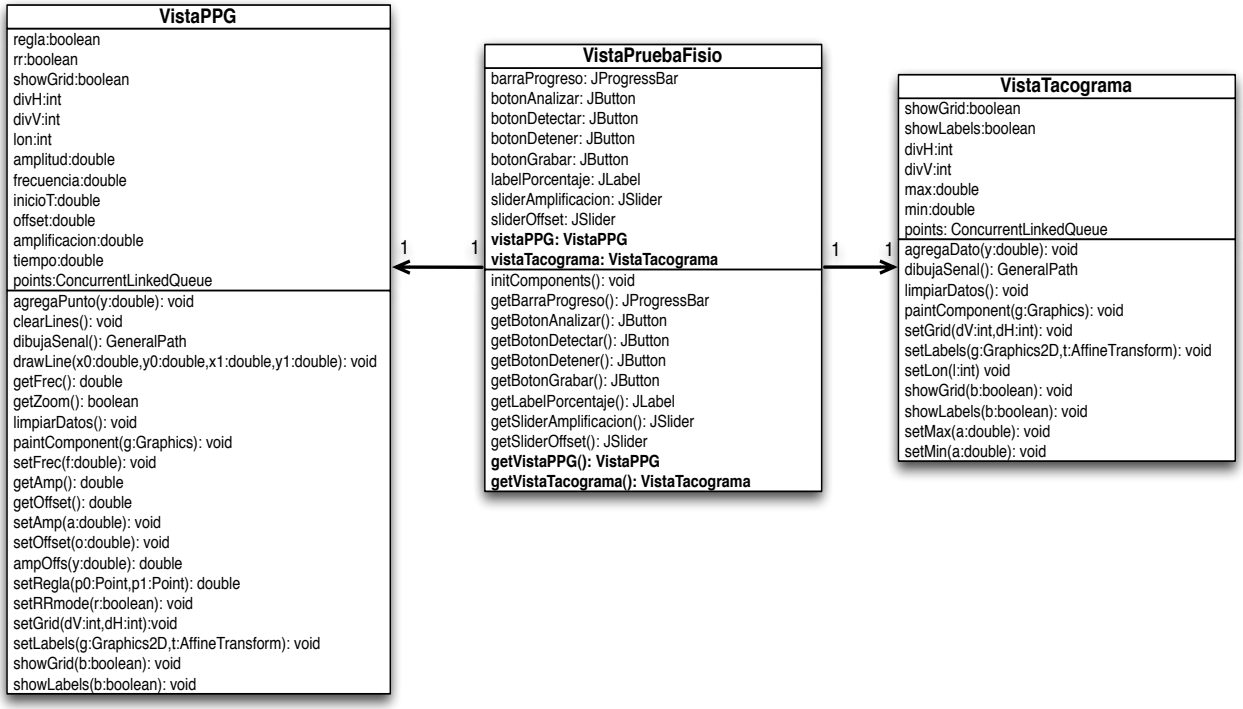

Figura 4.5: Diagrama de clases de la Interfaz gráfica

VistaPPG: Esta clase tiene como función principal graficar cada uno de los datos recibidos desde el dispositivo USB. En otras palabras, controla todos los aspectos relacionados a la visualización de la señal, ya sea antes y durante una grabación. Para ello implementa funciones que configuran el estado inicial de la visualización de la señal.

VistaTacograma: Esta clase entra en función justamente después del terminó de una grabación. Tiene como función dibujar el *Tacograma*, el cual es la gráfica más utilizada para representar los tiempos R-R respecto al número de pulsaciones del corazón en un determinado tiempo.

En la figura 4.6 se muestra la representación de un tacograma para una grabación, donde los intervalos de tiempo R-R medidos en milisegundos (ms) se agrupan de manera vertical, tomando en cuenta el mínimo y el máximo de ellos para delimitar el tacograma. En la parte horizontal se agrupan las pulsaciones del corazón, de manera que a cada pulso le corresponda un tiempo R-R. El número de pulsaciones varía de acuerdo a la persona que es medida, sin embargo, el bloque de grabación maneja un tiempo de 5 minutos por prueba, dado que es el tiempo suficiente para capturar la mayoría de las frecuencias con las que trabaja el corazón.

Figura 4.6: Representación de un tacograma

Finalmente, en la implementación, las clases que construyen la interfaz gráfica del bloque de grabación presentan la pantalla de la figura 4.7.

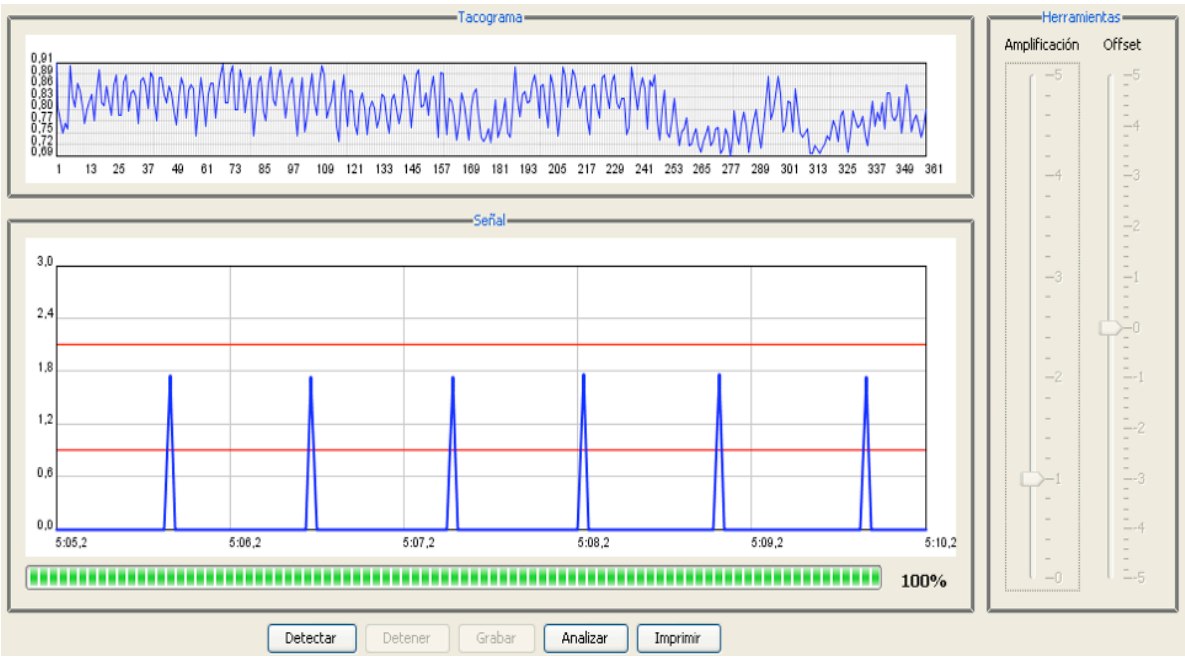

Figura 4.7: Interfaz gráfica del bloque de grabación

4.2.2 Interfaz de control del bloque de grabación

La interfaz de control está constituida por un conjunto de clases que controlan los eventos de la interfaz gráfica, calculan la derivada de la señal, obtienen y almacenan los tiempos R-R para su posterior análisis. Las clases que lo conforman se ilustran en el diagrama de clases de la figura 4.8.

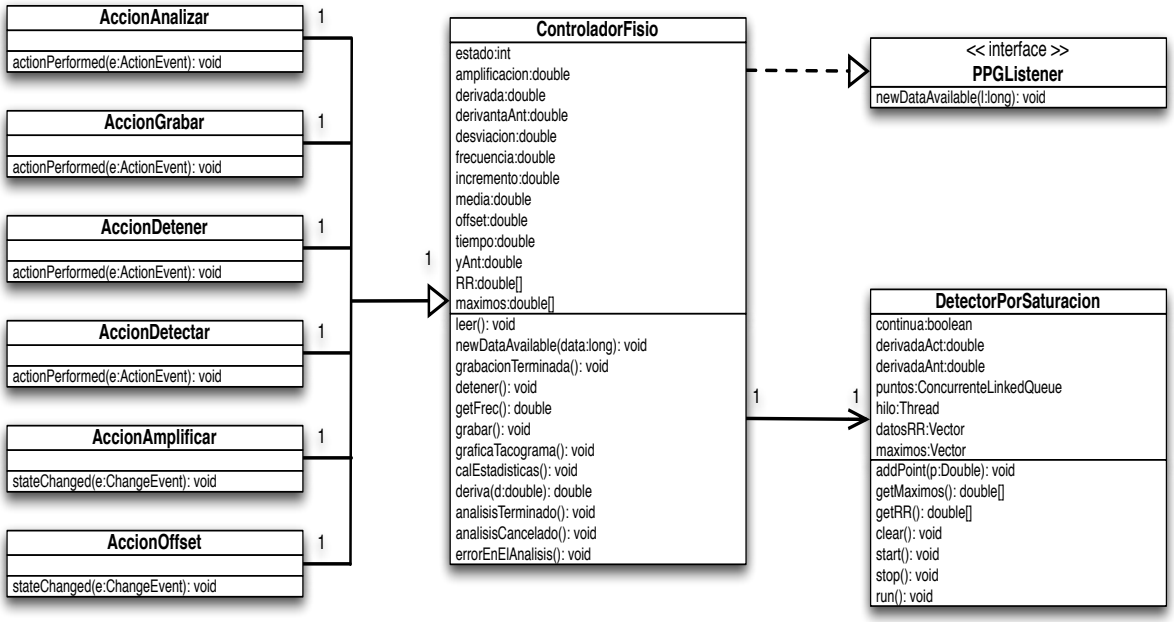

Figura 4.8: Diagrama de clases de la Interfaz de control

ControladorFisio: Esta clase interviene en todos los bloques del módulo de análisis y graficación, principalmente en los de grabación y análisis. En ella se implementan las principales funciones que llevan el control de cada etapa de una prueba de medición. Recepciona cada dato enviando por el módulo de transmisión para dos própositos; la primera es para graficar la señal y la segunda para el cálculo de la derivada de la misma. De igual forma, administra los eventos ocurridos en la interfaz gráfica; para esto, implementa las siguientes subclases:

- **AccionAnalizar**: Está directamente asociada al evento del objeto `botonAnalizar` de la interfaz gráfica, sin embargo también está relacionada al bloque de análisis iniciando su ejecución.

- **AccionGrabar**: Asociada al evento del objeto `botonGrabar` de la interfaz gráfica. Inicia el proceso de grabación al invocar la función `grabar()`.

- **AccionDetener**: Asociada al evento del objeto `botonDetener` de la interfaz gráfica. Tiene como función detener la grabación, así como la graficación de la señal.

- **AccionDetectar**: Asociada al evento del objeto `botonDetectar` de la interfaz gráfica. Inicia el proceso de graficación de la señal, o bien, la reinicia después de ser detenida invocando la función `leer()`.

- **AccionAmplificar**: Esta subclase está asociada a unos de los dos objetos de tipo *JSlider* que sirven para amplificar y acomodar la señal. En este caso se asocia al evento del objeto `sliderAmplificacion`.
- **AccionOffset**: Se asocia al evento del objeto `sliderOffset`, que tiene como función acomodar la señal en el panel de graficación.

PPGListener: Esta interface es muy importante, debido a que es la conexión entre el módulo de transmisión, con el módulo de análisis y graficación. En ella se declara la función `newDataAvailable()`, la cual recibe como parámetro los datos que el dispositivo de medición envía através del módulo de transmisión. Es decir, del lado de la transmisión está función es invocada, pero su implementación está en la clase `ControladorFisio`.

DetectorPorSaturacion: Esta clase desempeña un papel muy importante, tiene como función calcular la derivada de la señal, con la finalidad de obtener los tiempos R-R y posteriormente almacenarlos . Su implementación se basa en un hilo de ejecución controlado por la clase `ControladorFisio`.

Para detectar de los tiempos R-R, se implementó en el hilo de ejecución un algoritmo que constantemente verifica la entrada de nuevos datos para realizar el cálculo de la derivada. Este algoritmo se ha denominado *Detección por saturación* y su manera de procesar los datos se describe a continuación:

En primer lugar, la recepción de los datos desde la clase `ControladorFisio` a la clase `DetectorPorSaturacion` está dada por la función `addPoint`. La función `addPoint` recibe como parámetro una variable de tipo `double`. Esta variable contiene dos variables de tipo `long`, a una de ellas corresponde al punto de la señal y la otra al tiempo (`tiempo`) en que fue muestreado ese punto en la señal (`y`). La invocación e implementación de la función se muestra en el siguiente fragmento de código.

```
// Invocacion de addPoint () : ControladorFisio

addPoint(new Double(tiempo, y));

// Implementacion de addPoint() : DetectorPorSaturacion

public void addPoint(Double p){
   try
   {
            puntos.add(p);
   }
   catch(IllegalStateException e)
```

```
   {
      e.printStackTrace();
   }
}
```

En la implementación de la función `addPoint`, al recibir el valor de la variable `p`, la agrega a una variable de tipo `ConcurrentLinkedQueue` denominada como `puntos`. Una variable del tipo `ConcurrentLinkedQueue` nos permite crear una colección de elementos de diferentes tipos, en este caso se almacenan objetos del tipo `double`. Esta colección de elementos funciona como una lista, siendo el primer elemento que entra, será el primero que sale. De esta manera, lo puntos de la señal con sus respectivos tiempos son guardados en el orden de agregación para su posterior análisis. A partir de ahora no referimos esta colección como la colección de puntos.

El siguiente fragmento de código, constituye la parte principal del bloque de grabación, por ello a continuación se describen las líneas de código que permiten obtener los tiempos R-R mediante el cálculo de la derivada de la señal.

```
// Valores iniciales
// boolean continua = true
// double pActual = null
// double pAnterior = 0
// double derivadaAct = null
// double derivadaAnt = null
// double ultimoMax = null
// Vector datosRR
// Vector maximos

[1] public void run()
[2] {
[3]  do
[4]       while(!puntos.isEmpty())
[5]       {
[6]         pActual = puntos.poll();
[7]         if(pAnterior != null)
[8]         {
[9]          derivadaAct = pActual.getY() - pAnterior.getY();
[10]         if(derivadaAct < 0.0D && derivadaAnt >= 0.0D)
[11]         {
```

```
[12]           if(ultimoMax != null)
[13]             datosRR.add(new Double(pActual.x - ultimoMax.x));
[14]           ultimoMax = pActual;
[15]           maximos.add(new Double(ultimoMax.y));
[16]          }
[17]        }
[18]        pAnterior = pActual;
[19]        derivadaAnt = derivadaAct;
[20]      }
[21]   while(continua);
[22] }
```

1. El código expuesto anteriormente, corresponde al cuerpo principal del hilo de ejecución. Inicialmente se declaran los valores iniciales de las variables que intervienen en el algoritmo. Destacando entre esas variables las de tipo `Vector`, siendo el vector `datosRR` para almacenar los tiempos R-R obtenidos y el vector `maximos` para almacenar los puntos picos de la señal[1].

2. En la línea `[3]`, se presenta un ciclo `do-while` interactuando mientras que la variable `continua` (línea `[21]`) deje de ser verdadera. Dicha variable, es controlada por la clase `ControladorFisio` para detener el proceso de grabación.

3. La línea `[4]` verifica si la colección de puntos (`puntos`) este vacía.

4. En la línea `[6]` se extrae el primer punto de la colección, donde la variable `pActual` es de tipo `double` para poder ser compatible con las datos agregados a la colección inicialmente por la función `addPoint`. La variable `pActual` es denotada así por que representa al punto actual a ser analizado.

5. La línea `[7]` es una condicional para identificar si existe un previo punto analizado o bien, si es el primer punto en ser analizado; de ser así, el algoritmo se salta hasta la línea `[18]`. Para denotar al punto anterior analizado se utiliza la variable `pAnterior`.

6. Las líneas `[8]` y `[9]` son el par de instrucciones que definen el punto resultante después de calcular la derivada del punto actual, respecto al punto anterior. Pero antes, es necesario comprender que las variables `pActual` y `pAnterior` ambas del tipo `double`, están conformadas por dos datos de tipo `long`, el punto real de la señal y el tiempo en que fue muestreado, denotados como `pActual.y` y `pActual.x` respectivamente.

[1]Nos referimos como "punto pico" al punto más alto en la curvatura de la señal.

Dado lo anterior, en la línea `[8]` se realiza una asignación a la variable `derivadaAct` el resultado de la diferencia del punto actual con el punto anterior. Si el resultado de `derivadaAct > 0` indica que la señal no ha llegado a su punto más alto, en el cado de `derivadaAct < 0` la señal ya ha alcanzado el punto máximo. Por otro lado, la línea `[9]` añade una condicional para determinar el resultado o punto resultante de la derivada. De esta manera, entre las líneas `[8]` y `[9]` se presentan tres casos posibles en el transcurso del procediemiento.

Caso 1: Sucede cuando la señal no ha alcanzado su punto máximo, en este caso `derivadaAct > 0` y `derivadaAnt`$\geq$`0`. Como resultado de la condiciones se obtiene un punto de valor de `0`, debido que solo se desea obtener el punto más alto de cada curva de la señal, descartando los demás. En la figura 4.9 se muestra el caso 1, presentando el estado de las variables y posteriormente el resultado de la derivación.

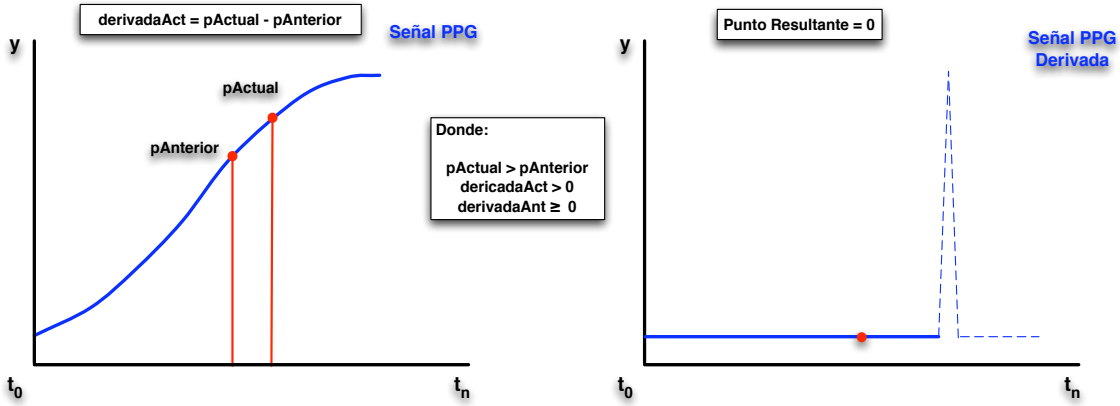

Figura 4.9: Caso 1 para el cálculo de la derivada

Caso 2: Este caso sucede cuando se ha alcanzado el punto más alto de la curva y los siguientes puntos empiezan a decrecer. Debido que `pActual < pAnterior`, entonces `derivadaAct`$<$`0` y si se cumple `derivadaAnt`$\geq$`0`, se obtiene un punto del mismo valor a `pAnterior`. Es decir, `pAnterior` ha sido el punto máximo de la curva. La figura 4.10 muestra el representación gráfica del caso 2.

Caso 3: Ocurre cuando la señal se encuentra en su punto más bajo y empieza crecer, indicando el inicio de una nueva curva. En este caso `pActual > pAnterior`, por lo tanto `derivadaAct > 0` y se cumple `derivadaAnt < 0` debido a que en la iteración anterior el punto actual era menor al punto anterior. Como resultado se obtiene un punto de valor `0`. Es entonces, cuando se vuelve a tener las condiciones del caso 1, y el ciclo se vuelve a repetir, hasta el momento que el

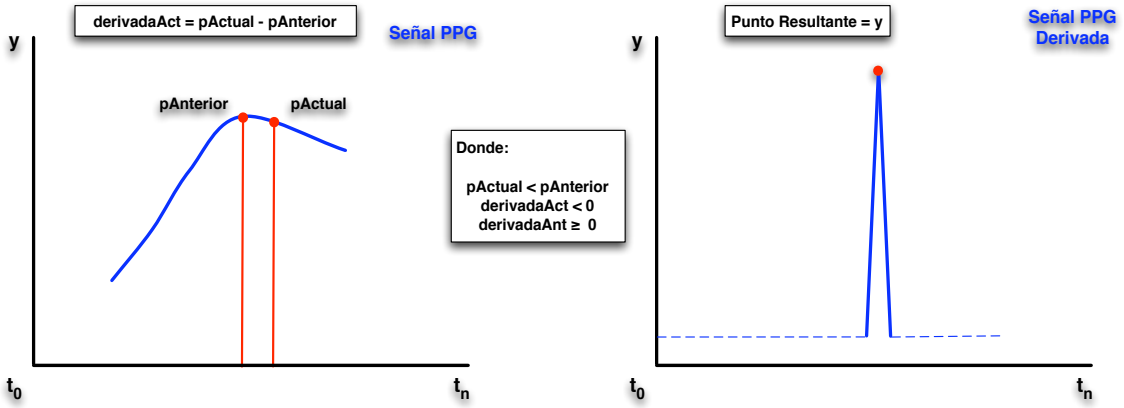

Figura 4.10: Caso 2 para el cálculo de la derivada

tiempo de la grabación se haya terminado. La representanción gráfica del caso 3 se muestra en la figura 4.11.

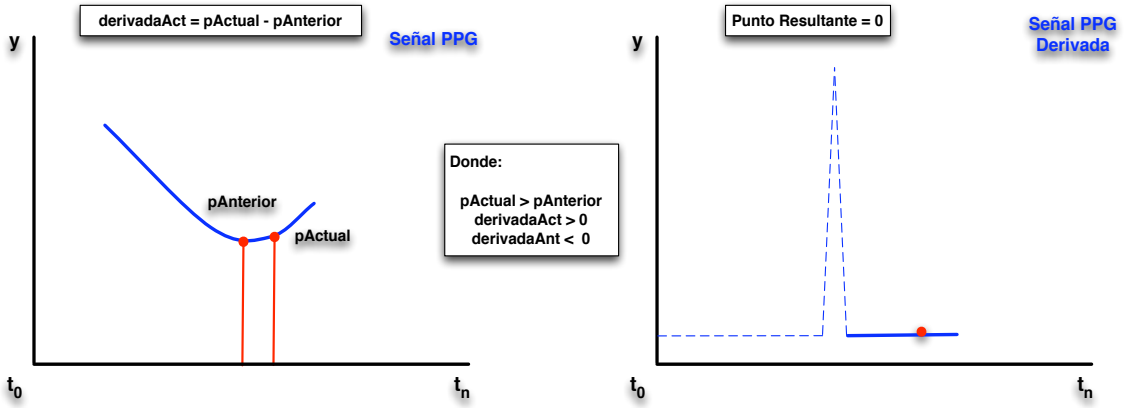

Figura 4.11: Caso 3 para el cálculo de la derivada

7. Posteriormente, al identificar el punto máximo de la curva, es necesario verificar si existe algún otro punto máximo anteriormente identificado (línea `[12]`), en caso de existir, se restan los tiempos de ambos puntos y se obtiene el tiempo R-R (línea `[13]`), es decir, el tiempo transcurrido entre la identificación del punto máximo anterior respecto al punto máximo actual. El tiempo R-R se almacena en el vector `datosRR`.

```
[12]  if(ultimoMax != null)
[13]    datosRR.add(new Double(pActual.x - ultimoMax.x));
[14]  ultimoMax = pActual;
[15]  maximos.add(new Double(ultimoMax.y));
```

8. Se actualiza el valor del último punto máximo (línea `[14]`) y se almacenan los puntos máximos en el vector `maximos` (línea `[15]`).

Finalmente, los vectores `datosRR` y `maximos` son el resultado las funciones del bloque de grabación, y ellos son de principal importancia, por que apartir de ellos se construye el tacograma de la interfaz gráfica y son los utilizados por el bloque de análisis; bloque que se describe en la siguiente sección.

4.3 Implementación del bloque de análisis de datos

El **bloque de análisis** es el encargado de realizar en análisis posterior a la obtención de los tiempos R-R, con la finalidad de encontrar las frecuencias altas y bajas de las pulsaciones cardíacas. Para ello, su implementación está dada por las clases mostradas en el diagrama de la figura 4.12.

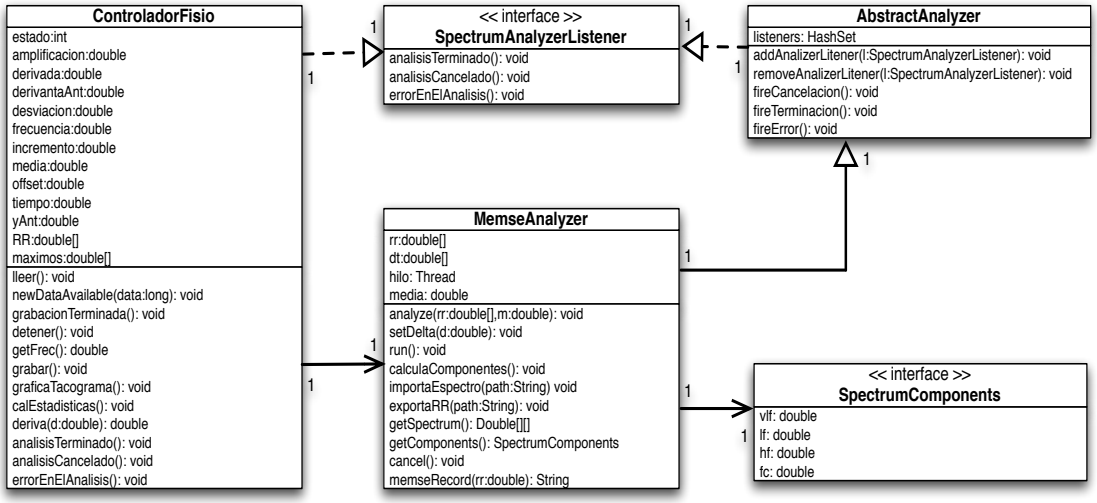

Figura 4.12: Diagrama de clases del bloque de análisis

ControladorFisio: Como en el bloque de grabación, esta clase controla el evento que inicia el proceso de análisis de los datos, en este caso, nos referimos al evento

controlado por la subclase `AccionAnalizar` definido en la sección 4.2.2. Otra función realizada por la clase previo al análisis, es la de transferir los datos del vector `datosRR` al arreglo `RR`. Dicho arreglo es de tipo `double` como se muestra en el diagrama 4.12. La transferencia de datos al arreglo mencionado, es necesaria debido que el método utilizado para el cálculo de frecuencias precisa de este tipo de arreglo.

Para iniciar el proceso de análisis, `ControladorFisio` instancía el objeto `analizador` de la clase `MemseAnalyzer` e invoca la función `analyze`, la cual recibe como párametros el arreglo `RR` y la variable `media`; esta última es la media de los tiempos R-R.

```
// analizador : objeto de la clase MemseAnalyzer
analizador.analyze(RR, media);
```

MemseAnalyzer: Está clase contiene todas las funciones encargadas del cálculo de frecuencias, sin embargo, la implementación de estas funciones, no quedó a cargo de nuestro trabajo, es decir, se obtuvieron a partir de otra fuente y solo fueron adaptadas a la necesidades de nuestra aplicación [45]. Lo anterior incluye a la clase `AbstractAnalyzer` y a las interfaces `SpectrumComponents` y `SpectrumAnalyzerListener`. Por otro lado, el método implementado en está clase es el método *Memse* o también conocido como el *método de máxima entropía (MEM)* [16, 46]. De ahí el nombre de la clase `MemseAnalyzer`.

Método Memse: Tiene como finalidad calcular el espectro de frecuencias correspondiente a una señal muestreada. En este caso la señal muestreada corresponde al tacograma representado por el arreglo `RR`. Su implementación se basa en un archivo ejecutable **memse.exe** proporcionado por [45]. El ejecutable recibe de entrada un archivo de texto y genera uno de salida, estos son:

- **ecgdata5mphysio.txt**: Es el archivo de entrada, el cual debe contener los tiempos R-R del arreglo `RR`. En este caso, la clase `MemseAnalyzer` implementa la función `exportaRR` encargada de la transferencia de los datos del arreglo `RR` al archivo *ecgdata5mphysio.txt*.
- **ecgdata5mmem.txt**: Es el archivo de texto generado por el ejecutable **memse.exe**, su contenido se conforma del espectro de frecuencias obtenido, es decir, contiene las todas las frecuencias encontradas de acuerdo al valor de los tiempos R-R de entrada. Sin embargo, el proceso no termina en este punto, por tal motivo la función `importaEspectro` de la clase `MemseAnalyzer`, importa los datos del archivo *ecgdata5mmem.txt* a un arreglo con nombre `spectrum`.

El siguiente paso del método, es tomar las frecuencias almacenadas en el arreglo `spectrum` y determinar a la banda de frecuencia que pertenece cada una. Las bandas

de frecuencias para este tipo de mediciones usualmente se dividen en tres con sus respectivos rangos [17, 47]. Estas bandas son las ilustradas en la figura 4.13.

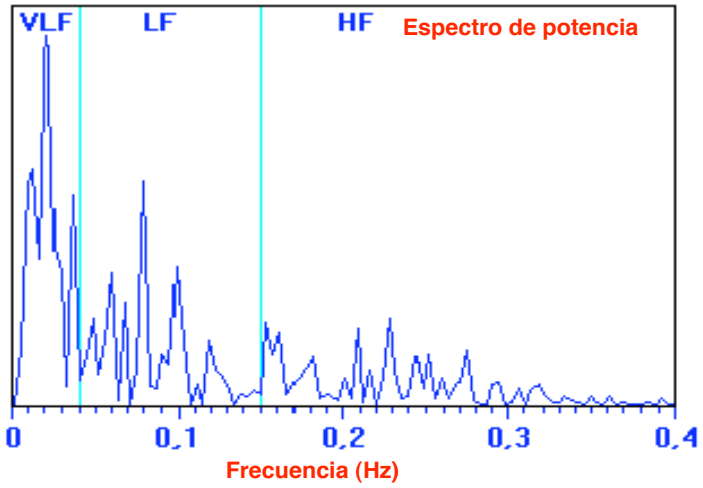

Figura 4.13: Bandas de frecuencia en el espectro

- Banda de muy baja frecuencia **(VLF)** para frecuencias inferiores a 0,04 Hz [20].
- Banda de baja frecuencia **(LF)** para frecuencias entre 0,04 Hz y 0,15 Hz [21].
- Banda de alta frecuencia **(HF)** para frecuencias entre 0,15 Hz y 0,4 Hz [22].

La fijación de los rangos de frecuencia no es aleatorio, ellos son el resultados de estudios que vinculan estas frecuencias con la interacción entre los sistemas SS y SP del sistema nervioso [18], sin embargo, en esta sección no entraremos a detalle sobre el tema, debido que se discutirá en el bloque de graficación de resultados.

La finalidad de definir las bandas de frecuencias, es para obtener las potencias de las frecuencias ubicadas en su respectivas bandas. De esta manera el resultado final del método *memse* son los tres valores correspondientes a: **VLF** la potencia total de las frecuencias muy bajas, **LF** la potencia total de las frecuencias bajas y **HF** la potencia total de las frecuencias altas. De igual forma, el valor de la frecuencia cardíaca **fc** obtenido al dividir 60 segundos entre la `media` de los tiempos R-R.

En este punto, el análisis ha finalizado, y la clase `MemseAnalyzer` invoca la función `fireTerminacion` incluida en la clase `AbstractAnalyzer` para indicar el termino satisfactorio del análisis, en caso contrario invoca la función `fireError`, o bien, la función `fireCancelación` si fue cancelado por un evento externo (usuario).

Interface SpectrumAnalyzerListener: Cuando finaliza el análisis, esta interface funciona como puente para comunicar a la clase `ControladorFisio` el estado de dicho proceso.

Interface SpectrumComponents: Esta interface es esencial, por ser la portadora de los resultados del bloque de análisis, necesarios para el bloque de graficación de resultados. Bloque el cual es discutido en la siguiente sección.

4.4 Implementación del bloque de graficación de resultados

El **bloque de graficación de resultados** como su nombre lo indica, es el encargado de gráficar y mostrar los resultados obtenidos del análisis. Pero antes de ello, tiene como función llevar acabo un diagnóstico en base a las potencias totales de las frecuencias **VLF, LF** y **HF**. Con la finalidad de cumplir con las funciones para las que fue destinado, el bloque está implementado en las clases del diagrama de la figura 4.14.

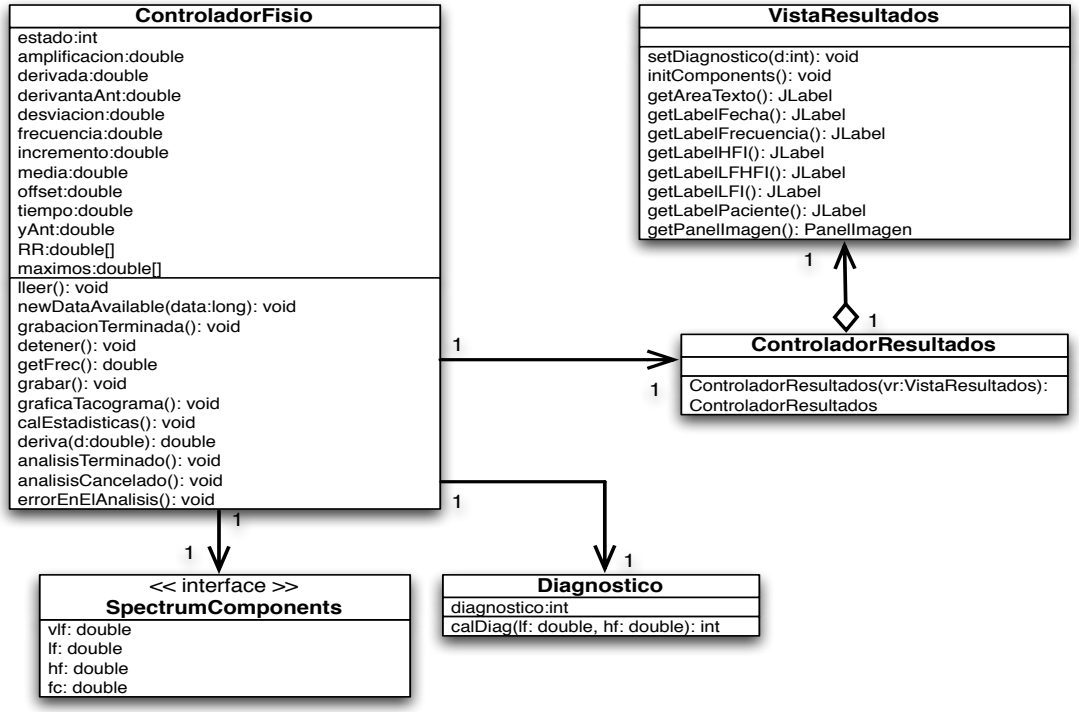

Figura 4.14: Diagrama de clases del bloque de graficación de resultados

ControladorFisio: Obtiene los valores de la potencia de las frecuencias, al instanciar un objeto de la interface `SpectrumComponents`. Posteriormente instancía un objeto de la `Diagnostico`, la cual recibe como parámetros los valores de **LF** y **HF**.

Diagnostico: Esta clase es la encargada de evaluar los valores arrojados por el bloque de análisis para determinar el nivel de estrés. La identificación del nivel de estrés, es

conforme a los resultados de las potencias de frecuencias de **VLF, LF** y **HF**, esto se debe a la asociación de cada banda de frecuencia a una función del sistema nervioso.

- **VLF**: Se asocia a las funciones del Sistema Nervioso Simpático.
- **LF**: Es asociado principalmente a la actividad del Sistema Nervioso Simpático.
- **HF**: Se asocia al Sistema Nervioso Parasimpático.

La asociación de las bandas de frecuencias a los sistemas simpático y parasimpático, permite establecer un conjunto de niveles de estrés dependiendo del balanceo de los sistemas SS y SP. Para definir los niveles del estrés, nos basamos en una proporción de balanceo entre los valores de **LF** y **HF**, es decir, si la diferencia entre ambos valores es cada vez mayor, se deduce que el estrés es más alto. En base a lo anterior, se definieron 11 niveles de estrés, que son representados como se muestra en la figura 4.15.

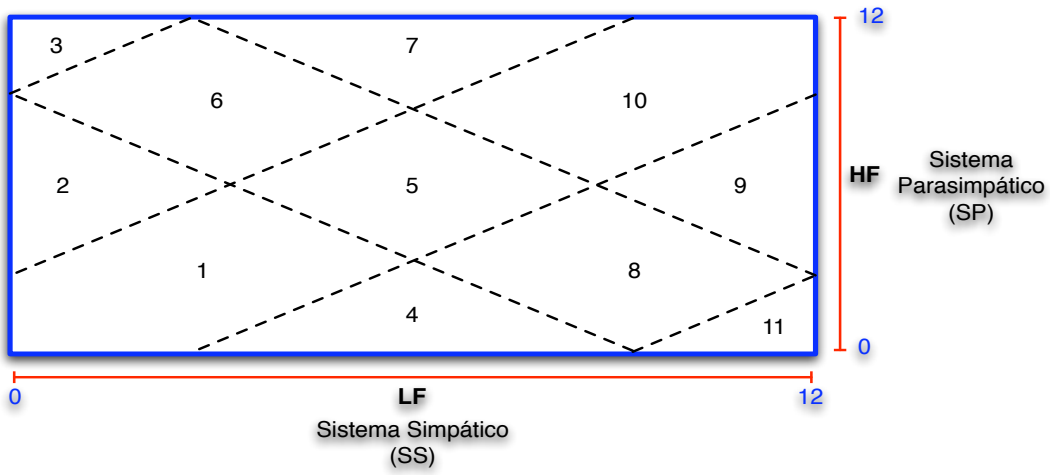

Figura 4.15: Diagrama de balanceo para los niveles de estrés

La manera de representar los niveles de estrés, está basada de acuerdo al presentado por [47], sin embargo, en dicha fuente solo definen 9 niveles sin utilizar el criterio de balanceo propuesta en nuestro trabajo.

En el diagrama de la figura 4.15, se visualizan 11 regiones y a cada una se le ha asignado un nivel de estrés. Donde, la parte horizontal es el valor de **LF**, y la parte vertical es el valor de **HF**, ambos valores son colocados en un rango de 0-12, simplemente para tener un mejor control de los valores al momento de ser graficados.

A continuación se definen las regiones con sus respectivas descripciones:

- **Región 1** (**LF**=Bajo, **HF**=Bajo): "Balanceo entre el SS y SP con nivel de actividad muy bajo de regulación por parte de ambos sistemas". Estrés moderado.

- **Región 2** (**LF**=Bajo, **HF**=Medio): "Desbalanceo entre el SS y SP con nivel bajo de actividad de regulación por parte del SS y nivel medio del SP". Estrés alto.
- **Región 3** (**LF**=Bajo, **HF**=Alto): "Peligro, sistema altamente No Balanceado con actividad de regulación dominante del SP". Estrés muy alto.
- **Región 4** (**LF**=Medio, **HF**=Bajo): "Desbalanceo entre el SS y SP con nivel de actividad media de regulación del SS y nivel bajo del SP". Estrés alto.
- **Región 5** (**LF**=Medio, **HF**=Medio): "Balanceo entre el SS y SP con nivel normal de actividad de regulación por parte de los dos sistemas". Estrés normal.
- **Región 6** (**LF**=Bajo, **HF**=Alto): "Desbalanceo entre el SS y SP con actividad baja de regulación del SS y nivel alto del SP". Estrés alto.
- **Región 7** (**LF**=Medio, **HF**=Alto): "Desbalanceo entre el SS y SP con nivel de actividad media del SS y nivel alto del SP". Estrés alto.
- **Región 8** (**LF**=Alto, **HF**=Bajo): "Desbalanceo entre el SS y SP con nivel alto de actividad de regulación del SS y nivel bajo del SP". Estrés alto.
- **Región 9** (**LF**=Alto, **HF**=Medio): "Desbalanceo entre el SS y SP con nivel de actividad alta de regulación del SS y nivel medio del SP". Estrés alto.
- **Región 10** (**LF**=Alto, **HF**=Alto): "Balanceo entre el SS y SP con un nivel alto de regulación por ambos sistemas". Estrés moderado.
- **Región 11** (**LF**=Alto, **HF**=Bajo): "Peligro, sistema altamente No Balanceado con actividad de regulación dominante del SS". Estrés muy alto.

Por la definición de estas regiones, es el motivo por el que podemos dar una interpretación a los valores de las frecuencias. Pero es necesario señalar que los niveles de estrés aquí presentados no son un estándar, hasta el momento en la literatura no existe un estándar para este tipo de mediciones. Por lo tanto, los niveles definidos en este proyecto son una propuesta más para ser tomados en cuenta en una posible estandarización.

4.4.1 Visualización de resultados

Por otro lado, el bloque de graficación, también se encarga de visualizar las regiones, el resultado del diagnóstico y los valores adquiridos durante el análisis. En este caso, son las clases `ControladorResultados` y `VistaResultados` encargadas de llevar acabo tales actividades.

ControladorResultados: Esta clase controla los aspectos de visualización de la pantalla de resultados. Coloca los resultados en sus componentes gráficos correspondientes.

VistaResultados: Esta clase contiene los componentes gráficos necesarios para presentar los resultados de manera adecuada. El diseño de la pantalla de resultados se divide en tres paneles.

- El *primer panel* le corresponde a los datos del paciente que realiza la medición, la fecha y los valores de los sistemas SS y SP, así también la frecuencia cardíaca (ver figura 4.16).

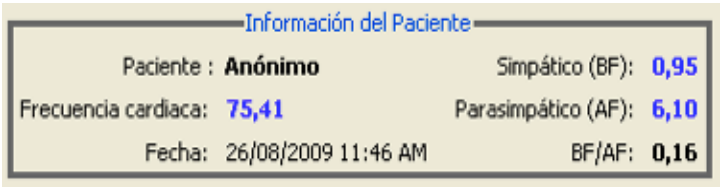

Figura 4.16: Panel 1: Resultados de los sistemas SS y SP e información del paciente

- El *segundo panel* contiene la descripción del diagnóstico obtenido, en otras palabras, nos referimos a la descripción de la región en donde los valores de las frecuencias **LF** y **HF** se interceptaron (ver figura 4.17).

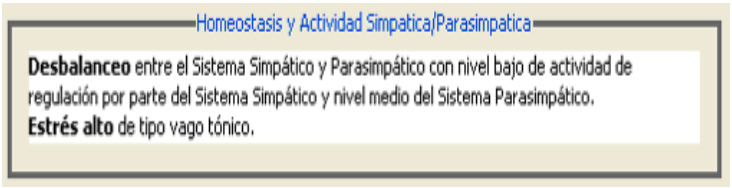

Figura 4.17: Panel 2: Descripción del diagnóstico de la prueba

- Al *tercel panel* le corresponde la visualización de la diagrama de balanceo de los sistemas SS y SP. En donde se indican con un punto, la región correspondiente al diagnóstico obtenido (ver figura 4.18).

Conjuntando los paneles en una pantalla, se obtiene la pantalla final de la figura 4.19, indicando la finalización de bloque de graficación de resultados, y por consiguiente el término satisfactorio de los procesos efectuados por el módulo de análisis y graficación.

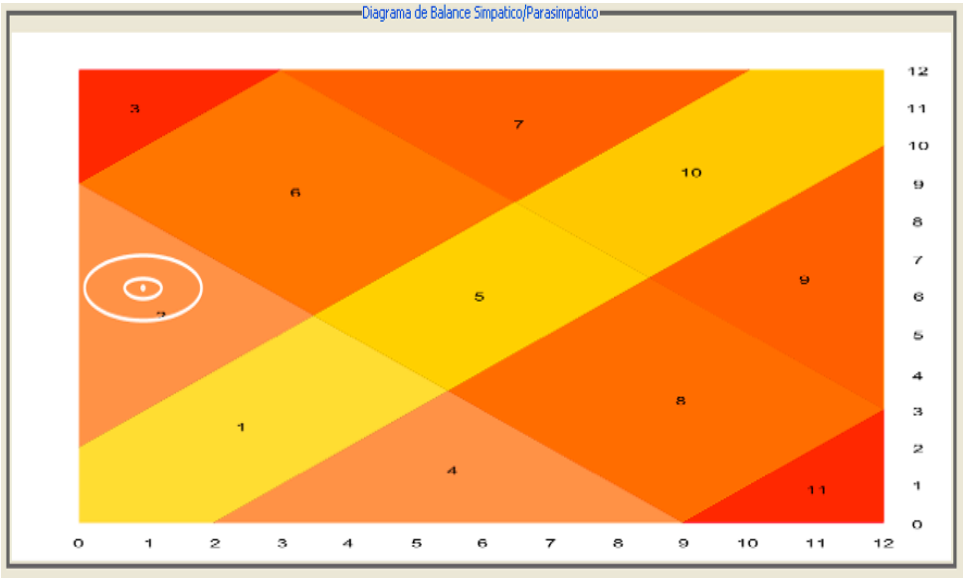

Figura 4.18: Panel 3: Diagrama de balanceo de los sistemas SS y SP

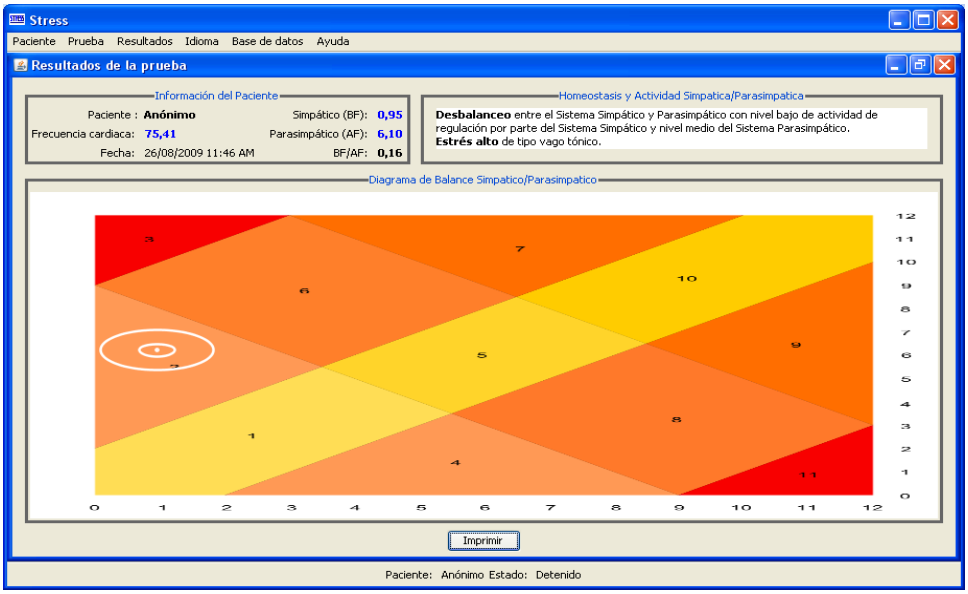

Figura 4.19: Pantalla final del bloque de graficación de resultados

Capítulo 5

Pruebas y resultados

Después de haber descrito los módulos de transmisión, análisis y graficación de datos, este capítulo se enfoca en las pruebas realizadas al sistema y de la descripción de los resultados obtenidos.

Para realizar las pruebas pertinentes, fue necesario determinar el mejor caso de estudio que nos perimitiera evaluar la eficiencia del sistema.

5.1 Caso de estudio

El campo del estudio del estrés ha sido estudiado desde hace mucho tiempo, a pesar que ahora sabemos de sus influencias sobre la salud de una persona, la medición del estrés ha tomado ciertas discrepancias en torno a la técnica más adecuada para determinar un medida cualitativa o cuantitativa.

Estas diferencias han llevado a la inexistencia de un estándar de medición para el estrés, por lo tanto, comparar nuestro sistema con algún otro parecido sería de poca relevancia, debido a las diferentes métricas utilizadas para las mediciones.

Por ello, las pruebas realizadas al sistema, se basaron en la medición de un grupo de personas. Esto nos permitió evaluar el desempeño de la conexión, transmisión, análisis y graficación del sistema.

5.2 Infraestructura

El equipo de computó utilizado para llevar acabo las pruebas tiene las siguientes características:

- Laptop MacBook Pro, Procesador Intel Core Duo 2.6 Ghz, Memoria Ram 2GB. Sistema Operativo Windows XP Profesional.
- Laptop Compaq Presario, Procesador AMD Sempron 2.0 Ghz, Memoria Ram 1GB, Sistema Operativo Windows Vista Home Basic.

Se probaron en ambos equipos, para determinar la funcionalidad del sistema en las diferentes plataformas de Windows.

5.3 Pruebas

El sistema fue instalado en los equipos de cómputo antes mencionados. En ellas se realizaron las mediciones a personas y en a base a ellas se tomaron en cuentas las siguientes pruebas:

- **Pruebas de instalación**: Se realizó la instalación del sistema en ambas plataformas (Windows XP y Vista), verificando que los archivos necesarios se hayan copiado en el archivos correspondientes.
- **Pruebas de conexión del dispositivo**: Se probó que el dispositivo fuera reconocido por el SO como un dispositivo USB.
- **Pruebas de conexión dispositivo/sistema**: Esta prueba consistió en verificar si el sistema iniciaba correctamente con el dispositivo reconocido.
- **Pruebas de transmisión**: Probamos la transmisión de lo datos, verificando que la señal recibida fuera la adecuada.
- **Pruebas de funcionalidad**: Se probó el flujo de datos fueran correctos, así como el análisis de la señal obtenida y la graficación de resultados en las mediciones fisiológicas realizadas.

5.4 Resultados

Las siguientes secciones se describen los resultados obtenidos de las pruebas realizadas individualmente al sistema. Dichos resultados se basan en la estadística generada por el número de mediciones llevadas acabo a diferentes personas.

5.4.1 Instalación del sistema

La instalación del sistema fue satisfactoria para ambas plataformas, copiando los archivos en el directorio correspondiente. De la misma manera se crearon los accesos directos correspondientes para el fácil acceso al sistema.

5.4.2 Conexión Dispositivo/SO

Por parte de las pruebas realizadas al dispositivo se comprobó la conexión entre el SO y el dispositivo USB de manera adecuada. Donde el dispositivo USB era reconocido como tal por parte del SO, con lo que se comprobó el buen funcionamiento del FW y el controlador instalado.

5.4.3 Conexión Dispositivo/Sistema

La conexión entre dispositivo y nuestro sistema, fue del todo exitosa, debido a la mejora realizada entre el sistema anterior y el ahora actualizado en cuanto a la velocidad. Anteriormente el sistema debía cargar en memoria ciertas bibliotecas necesarias para llevar acabo la emulación del puerto USB y se convertía en un proceso de aproximadamente 60 segundos. Con la actualización, el tiempo de conexión se redujó a pocos segundos. La pantalla de conexión se muestra en la figura 5.1.

Figura 5.1: Conexión entre el dispositivo USB y el Sistema

5.4.4 Transmisión de datos

La transmisión de datos utilizando el protocolo USB fue satisfactoria en todos los casos, al enviar la señal digitalizada de forma correcta. Esto se comprobó al observar la señal característica de la técnica de adquisición de *fotopletismografía* (ver figura 5.2).

Como prueba adicional, se creó un programa que permitiera imprimir en consola los bytes recibidos desde del dispositivo; esto se ve en la figura 5.3. Ambas pruebas demostraron que la implementación del módulo de transmisión USB 2.0 fue acertada y funciona correctamente.

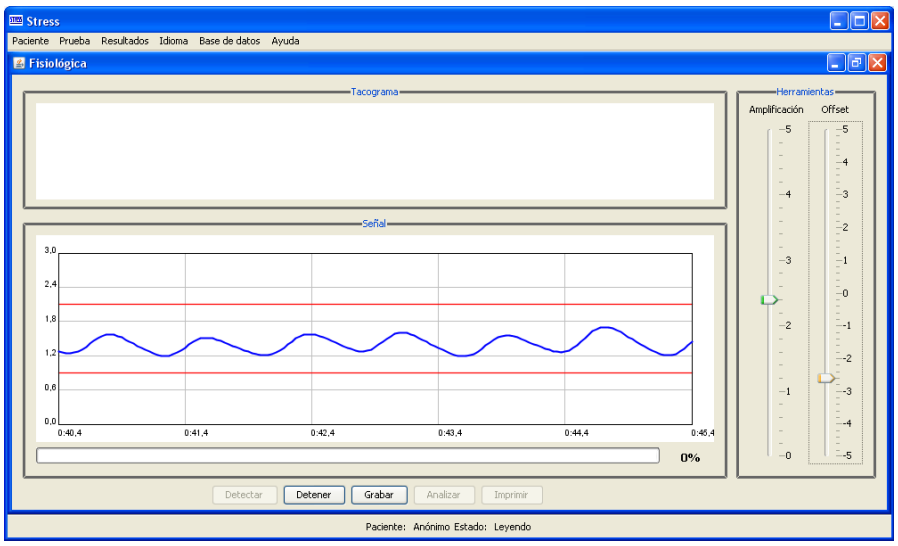

Figura 5.2: Transmisión de la señal por medio del protocolo USB 2.0

Figura 5.3: Recepción de bytes desde el dispositivo USB

Como dato extra, en algunos de casos la transmisión de la señal se mostró débil, esto ocurrió con personas que tenían el pulso débil; para ello el sistema implementa un amplificador con la finalidad de obtener una mejor visualización de la señal.

5.4.5 Funcionalidad del Sistema

El sistema muestra una correcta visualización de los componentes gráficos, botones, ventanas, etc. Por otra parte, para la graficación de la señal, el sistema presentaba un

correcto funcionamiento graficando la señal antes y durante la grabación de los datos provenientes del dispositivo USB.

La graficación de la señal durante el proceso de grabación se muestra en la pantalla de la figura 5.4, donde claramente se visualizan los valores picos de la señal.

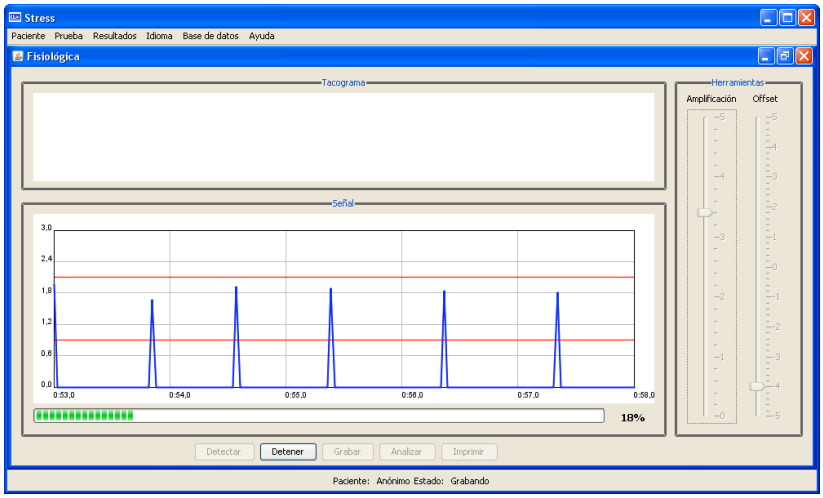

Figura 5.4: Graficación de la señal derivada

Se obtuvieron grabaciones completas de 5 minutos, almacenando los tiempos R-R adecuadamente e indicando la finalización de la grabación para su posterior análisis, como se ve en la figura 5.5.

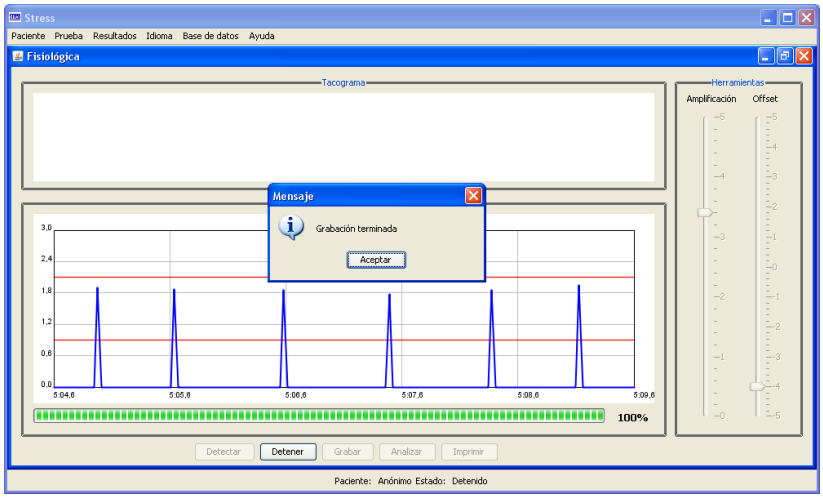

Figura 5.5: Indicación de una grabación completada satisfactoriamente

Los análisis de los datos resultaron ser eficientes, el sistema computa los datos obtenidos mediante el procedimiento desarrollado y posteriormente muestra el diagnóstico en la gráfica de niveles de estrés. La graficación del diagnóstico arrojado por el sistema, se ilustra en la figura 5.6.

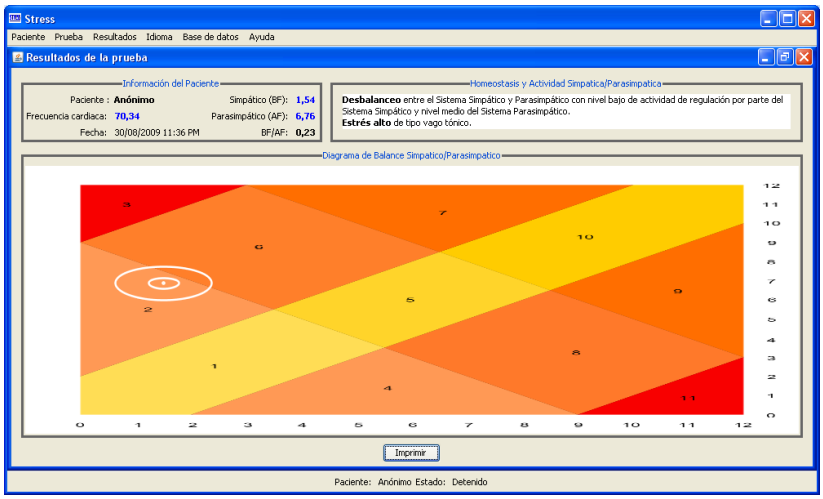

Figura 5.6: Graficación del diagnóstico de una prueba fisiológica

Capítulo 6

Conclusiones y trabajo futuro

6.1 Conclusiones

Finalmente se obtuvo sistema capaz de realizar un diagnóstico del los niveles estrés a través de la obtención de una señal fisiológica. Del trabajo realizado podemos concluir lo siguiente:

- El sistema es capaz de enviar los datos utilizando el protocolo USB 2.0, sin la necesidad de la emulación de una conexión serial.
- La utilización de protocolo USB 2.0 como el protocolo de comunicación entre sistema/dispositivo precede un importante ventaja por ser un estándar universal.
- La velocidad de conexión entre el sistema y el dispositivo USB es mucho más rápida en comparación a la conexión del sistema anterior.
- El uso de la API WinUSB en combinación con la JNI permiten al sistema ser compatible con los sistemas operativos Windows (XP y posteriores). Logrando eliminar la dependencia de bibliotecas necesarias para realizar la comunicación sistema/dispositivo como sucedía en la emulación del puerto serial.
- La velocidad de muestro se aumentó a 500 bytes/s, es decir, el sistema recibe 500 puntos por segundo del dispositivo USB. Esta velocidad se puede modificar a una mayor, pero 500 bytes/s era más que suficiente para este proyecto.
- El FW es ahora más fácil de modificar para trabajos posteriores, tanto en su programación como en la configuración del dispositivo USB.
- El sistema es capaz de realizar un análisis de los datos, calculando los tiempos R-R, obteniendo su rango de frecuencias y mostrando un diagnóstico final.
- La graficación del diagnóstico en un diagrama de niveles, ayuda al usuario del sistema a entender mejor el resultado de la medición.

6.2 Trabajo a futuro

Como trabajo futuro para este proyecto, existen líneas de trabajo interesantes, estas son algunas de ellas:

- Implementar un módulo de medición de la presión arterial, introduciendo una señal fisiológica más al dispositivo. Esta señal sería un electrocardiograma, la cual en combinación con señal obtenida del dedo, se puede calcular la presión arterial en base a una relación ya definida en otros estudios de esta índole [23, 24].
- Añadir un módulo de medición de glucosa en la sangre. Tanto trabajos clínicos y experimentales, han demostrado que el estrés es factor en las variaciones del nivel de glucosa en la sangre de una persona, por lo tanto, a través de una investigación experimental se puede crear un módulo de medición con la función de estimar el nivel de glucosa conforme a una medición del nivel de estrés [25, 26].
- Modificar el controlador del dispositivo USB para la compatibilidad con varios sistemas operativos. Una buena opción sería el uso de la API LibUSB, la cual es una biblioteca en desarrollo para la compatibilidad de dispositivos USB entre los sistemas Windows, Linux y Mac OS X.
- Modificar la electrónica del dispositivo al reducir su tamaño utilizando un microcontrolador de la misma familia al del pic18F4550, pero de menor tamaño.
- La adaptación del sistema de medición a dispositivos móviles por medio de la conexión usb. Para ello habría de utilizarse un microcontrolador que funcionará como *host* y cliente al mismo tiempo, esto debido que los sistemas operativos para dispositivos móviles no funcionan como *host*.

Apéndice A

A.1 Características generales del protocolo USB

La especificación del USB proporciona una serie de características que pueden ser distribuidas en categorías. Estas características son comunes para todas las versiones (desde la 1.0 hasta la 2.0).

Fácil uso para los usuarios

- Modelo simple para el cableado y los conectores.
- Detalles eléctricos aislados del usuario (terminaciones del bus).
- Periféricos auto-identificativos.
- Periféricos acoplados y reconfigurados dinámicamente (Hot Swappable).

Flexibilidad

- Amplio rango de tamaños de paquetes, permitiendo variedad de opciones de buffering de dispositivos.
- Gran variedad de tasas de datos de dispositivos acomodando el tamaño de buffer para los paquetes y las latencias.
- Control de flujo para el manejo del buffer construido en el protocolo.

Ancho de banda isócrono

- Se garantiza un ancho de banda y bajas latencias apropiadas para telefonía, audio, etc.
- Cantidad de trabajo isócrono que puede usar el ancho de banda completo del bus.
- Control de flujo para el manejo del buffer construido en el protocolo.

Amplia gama de aplicaciones y cargas de trabajo

- Adecuando el ancho de banda desde unos pocos kbps hasta varios Mbps.
- Soporta tanto el tipo de transferencia isócrono como el asíncrono sobre el mismo conjunto de cables.
- Conexiones múltiples, soportando operaciones concurrentes de varios dispositivos.
- Soporta hasta 127 dispositivos físicos.
- Soporta la transferencia de múltiples datos y flujos de mensajes entre el host y los dispositivos.

Robustez

- Manejo de errores y mecanismos de recuperación ante fallos implementados en el protocolo.
- Inserción dinámica de dispositivos.
- Soporte para la identificación de dispositivos defectuosos.

Implementación de bajo coste

- Subcanal de bajo coste a 1.5 Mbps.
- Conectores y cables de bajo coste.
- Adecuado para el desarrollo de periféricos de bajo coste.

A.2 Flujo de datos del protocolo USB

Un dispositivo USB desde un punto de vista lógico hay que entenderlo como una serie de *endpoints*, a su vez los *endpoints* se agrupan en conjuntos que dan lugar a interfaces, las cuales permiten controlar la función del dispositivo.

Como ya se ha visto la comunicación entre el host y un dispositivo físico USB se puede dividir en tres niveles o capas. En el nivel más bajo el controlador de host USB se comunica con la interfaz del bus utilizando el cable USB, mientras que en un nivel superior el software USB del sistema se comunica con el dispositivo lógico utilizando la tubería de control por defecto (*"Default Control Pipe"*). En lo que al nivel de función se refiere, el software cliente establece la comunicación con las interfaces de la función a través de tuberías asociadas a *endpoints*.

A.2.1 *Endpoints* y direcciones de dispositivo

Cada dispositivo USB está compuesto por una colección de endpoints independientes, y una dirección única asignada por el sistema en tiempo de conexión de forma dinámica. A su vez cada *endpoint* dispone de un identificador único dentro del dispositivo al que pertenece, a este identificador se le conoce como número de *endpoint* y viene asignado de fábrica. Cada *endpoint* tiene una determinada orientación de flujo de datos. La combinación de dirección, número de *endpoint* y orientación, permite referenciar cada *endpoint* de forma inequívoca. Cada *endpoint* es por si solo una conexión simple que soporta un flujo de datos en una única dirección, bien de entrada o bien de salida.

Cada *endpoint* se caracteriza por:

- Frecuencia de acceso al bus requerida.
- Ancho de banda requerido.
- Número de endpoint.
- Tratamiento de errores requerido.
- Máximo tamaño de paquete que el *endpoint* puede enviar o recibir.
- Tipo de transferencia para el *endpoint*.
- La orientación en la que se transmiten los datos.
- Existen dos *endpoints* especiales que todos los dispositivos deben tener, los *endpoints* con número 0 de entrada y de salida, que deben de implementar un método de control por defecto al que se le asocia la tubería de control por defecto. Estos *endpoints* están siempre accesibles mientras que el resto no lo estarán hasta que no hayan sido configurados por el *host*.

A.2.2 Tuberías

Una tubería USB es una asociación entre uno o dos *endpoints* en un dispositivo, y el software en el *host*. Las tuberías permiten mover datos entre software en el host, a través de un buffer, y un *endpoint* en un dispositivo. Hay dos tipos de tuberías:

- **Stream**: Los datos se mueven a través de la tubería sin una estructura definida.
- **Mensaje**: Los datos se mueven a través de la tubería utilizando una estructura USB.

Además una tubería se caracteriza por:

- Demanda de acceso al bus y uso del ancho de banda

- Un tipo de transferencia
- Las características asociadas a los *endpoints*

La tubería que está formada por dos *endpoints* con número cero se denomina tubería de control por defecto. Esta tubería está siempre disponible una vez se ha conectado el dispositivo y ha recibido un reseteo del bus. El resto de tuberías aparecen después que se confígure el dispositivo. La tubería de control por defecto es utilizada por el software USB del sistema para obtener la identificación y requisitos de configuración del dispositivo y para configurar al dispositivo.

Streams: No necesita que los datos se transmitan con una cierta estructura. Las tuberías stream son siempre unidireccionales y los datos se transmiten de forma secuencial: *"first in, first out"*. Están pensadas para interactuar con un único cliente, por lo que no se mantiene ninguna política de sincronización entre múltiples clientes en caso de que así sea. Un stream siempre se asocia a un único *endpoint* en una determinada orientación.

Mensajes: A diferencia de lo que ocurre con los streams, en los mensajes la interacción de la tubería con el *endpoint* consta de tres fases. Primero se realiza una petición desde el host al dispositivo, después se transmiten los datos en la dirección apropiada, finalmente un tiempo después se pasa a la fase estado. Para poder llevar a acabo este paradigma es necesario que los datos se transmitan siguiendo una determinada estructura. Las tuberías de mensajes permiten la comunicación en ambos sentidos, de hecho la tubería de control por defecto es una tubería de mensajes. El software USB del sistema se encarga de que múltiples peticiones no se envíen a la tubería de mensajes concurrentemente. Un dispositivo ha de dar únicamente servicio a una petición de mensaje en cada instante por cada tubería de mensajes. Una tubería de mensajes se asocia a un par de *endpoints* de orientaciones opuestas con el mismo número de *endpoint*.

A.2.3 Frames y microframes

USB establece una unidad de tiempo base equivalente a 1 milisegundo denominada frame y aplicable a buses de velocidad media o baja, en alta velocidad se trabaja con microframes, que equivalen a 125 microsegundos. Los microframes no son más que un mecanismo del bus USB para controlar el acceso a este, en función del tipo de transferencia que se realice. En un microframe se pueden realizar diversas transacciones de datos.

Apéndice B

B.1 Transferencias

La interpretación de los datos que se transmitan a través de las tuberías, independientemente de que se haga siguiendo o no una estructura USB definida, corre a cargo del dispositivo y del software cliente. No obstante, USB proporciona cuatro tipos de transferencia de datos sobre las tuberías para optimizar la utilización del bus en función del tipo de servicio que ofrece la función.

B.1.1 Transferencias de control

Las transferencias de control proporcionan control de flujo y una entrega de datos garantizada y libre de errores. Todos los dispositivos *full*, *high* y *low-speed* pueden incorporar *endpoints* de control, y por lo tanto pueden hacer uso de las transferencias de control. Todos implementan, al menos, un *endpoint* de salida y uno de entrada en la dirección 0, para poder establecer la tubería de control por defecto.

Las transferencias de control se componen de 3 transacciones denominadas Setup-Dato-Estado. Los tamaños máximos del paquete de datos durante la transacción de datos son:

- *Full-speed*: 8, 16, 32, o 64 bytes.
- *High-speed*: 64 bytes.
- *Low-speed*: 8 bytes.

USB hace una gestión *"best effort"* para ir dando curso a las distintas transferencias de control pendientes en cada momento en todas las tuberías de control establecidas con todos los dispositivos. Para ello se hace la siguiente reserva del tiempo del frame o microframe:

En un bus *full/low-speed*, la reserva es del 10% del tiempo del frame.

En un bus *high-speed*, la reserva es del 20% del tiempo del microframe.

Las reglas definidas por USB para el envío de las transferencias pendientes son:

- Si el tiempo del frame o microframe utilizado por las transferencias de control pendientes es inferior al reservado, el tiempo restante puede utilizarse para transferencias *Bulk*.
- Si hay más transferencias de control pendientes que tiempo reservado, pero hay tiempo adicional en el frame o microframe no consumido por transferencias de interrupción o isócronas, entonces el *host* puede utilizar dicho tiempo adicional para enviar nuevas transferencias de control.
- Si hay más transferencias de control pendientes que tiempo disponible en un frame o microframe, el *host* selecciona cuáles se procesan, quedando el resto pendientes para un próximo frame o microframe.

Los *enpoints* de control *high-speed* soportan el protocolo de control de flujo PING en las transacciones de datos y estado de salida.

B.1.2 Transferencias isócronas

Las transferencias isócronas están diseñadas para soportar aquellos dispositivos que precisan una entrega de datos a velocidad constante, y en la que no importa la pérdida eventual de información. Esto es necesario para aplicaciones en que la información de tiempo va implícita en la propia velocidad de transmisión/recepción de datos.

Para ello, las transferencias Isócronas proporcionan:

- Ancho de banda garantizado.
- Latencia limitada.
- Velocidad de transferencia de datos constante garantizada a través de la tubería.
- En caso de error en la entrega, no se reintenta la transmisión.
- Sin control de flujo.
- Sólo los dispositivos *high* y *full-speed* pueden incorporar *endpoints* isócronos.

Las transferencias Isócronas se componen sólo de transacciones de datos. Las frecuencias y los tamaños de los paquetes de datos son:

- *Full-speed*: 1 transacción por frame de hasta 1,023 bytes.
- *High-speed*: 1 transacción por microframe de hasta 1,024 bytes.
- *High-speed high-bandwidth*: 2 o 3 transacciones por microframe de hasta 1,024 bytes cada uno.

La gestión que hace USB para garantizar las transferencias es la de establecer o no la tubería en función de que haya suficiente tiempo libre del frame o microframe para realizarlas. Para ello, los *endpoints* isócronos indican qué cantidad de información como máximo debe transferir la tubería en cada frame o microframe, de forma que el sistema USB puede calcular si hay suficiente tiempo o no para acomodar la tubería, y en función de eso la establece o no.

La reserva de tiempo del frame o microframe para acomodar transferencias isócronas y de interrupción es como máximo el tiempo no reservado para transferencias de control. El sistema USB puede ir estableciendo tuberías isócronas y de interrupción con distintos dispositivos hasta agotar dicha reserva:

- *Full-speed*: Hasta un 90% del tiempo del frame.
- *High-Speed*: Hasta un 80% del tiempo del microframe.

B.1.3 Transferencias de interrupción

Las transferencias de Interrupción están diseñadas para soportar aquellos dispositivos que precisan enviar o recibir datos de manera no frecuente, pero con ciertos límites de latencia.

Para ello, las transferencias de Interrupción proporcionan:

- Tiempo máximo de servicio (latencia) garantizado.
- Reintento de transferencia en el siguiente periodo, en caso de eventual fallo en la entrega.
- Todos los dispositivos *high, full* y *low-speed* pueden incorporar *endpoints* de interrupción.

Las transferencias de Interrupción se componen sólo de transacciones de datos. Los tamaños de los paquetes de datos son:

- *Low-speed*: hasta 8 bytes.
- *Full-speed*: hasta 64 bytes.
- *High-speed*: hasta 1,024 bytes.
- *High-speed high-bandwidth*: 2 ó 3 transacciones por microframe de hasta 1,024 bytes cada uno.

La gestión que hace USB para garantizar las transferencias es la de establecer o no la tubería en función de que haya suficiente tiempo libre del frame o microframe para realizarlas. Para ello, las transferencias de interrupción indican qué cantidad de información como máximo debe transferir la tubería en cada transacción, así como el

tiempo máximo entre transacciones, de forma que el sistema USB puede calcular si hay suficiente tiempo o no para acomodar la tubería, y en función de eso la establece o no.

El tiempo máximo entre transacciones (tiempo de latencia máximo) especificado por cada dispositivo puede ser:

- *Low-speed*: de 10 a 255 ms.
- *Full-speed*: de 1 a 255 ms.
- *High-speed*: de 125 μs a 4'096 seg.

La reserva de tiempo del frame o microframe para acomodar transferencias isócronas y de interrupción es como máximo el tiempo no reservado para transferencias de control. El sistema USB puede ir estableciendo tuberías isócronas y de interrupción con distintos dispositivos hasta agotar dicha reserva:

- *Full* y *Low-speed*: Hasta un 90% del tiempo del frame.
- *High-Speed*: Hasta un 80% del tiempo del microframe

B.1.4 Transferencias bulk

Las transferencias bulk están diseñadas para soportar aquellos dispositivos que precisan enviar o recibir grandes cantidades de datos, con latencias que pueden tener amplias variaciones, y en los cuales, las transacciones pueden utilizar cualquier ancho de banda disponible.

Para ello, las transferencias *Bulk* proporcionan:

- Acceso al bus en función del ancho de banda disponible.
- Reintento de transferencias en caso de errores de entrega.
- Entrega garantizada de datos, pero sin garantía de latencia máxima ni de ancho de banda.

Las transferencias *Bulk* se realizan relativamente rápidas, si el bus dispone de mucho ancho de banda libre, pero en un bus USB con mucho ancho de banda reservado, pueden alargarse durante periodos de tiempo relativamente grandes. Sólo los dispositivos *high* y *full-speed* pueden incorporar *endpoints Bulk*.

Las transferencias *Bulk* se componen sólo de transacciones de datos. Los tamaños máximos de los paquetes de datos son:

- *Full-speed*: 8,16, 32 y 64 bytes.
- *High-speed*: 512 bytes.

USB gestiona el curso de las distintas transferencias pendientes en cada momento en todas las tuberías *Bulk* establecidas con todos los dispositivos. Las transferencias de control tienen preferencia sobre las *Bulk*, por lo que las transferencias *Bulk* se realizan siempre que no haya otro tipo de transferencias que hacer en un frame o microframe. Los *endpoints Bulk-OUT high-speed* soportan el protocolo de control de flujo PING.

Apéndice C

C.1 Arquitectura de WinUSB

En la figura C.1 ilustra una pila de controladores USB que contiene varios ejemplos de *Winusb.sys*. Esta figura muestra un ejemplo de configuración WinUsb que implementa tres clases de interfaz de dispositivo, cada uno de los cuales tiene una única interfaz de dispositivos registrados:

- Caso 1 de *winusb.sys*: Registra la interfaz del dispositivo A, que admite un controlador en modo de usuario (*Usboem.dll*).

- Caso 2 de *winusb.sys*: Registra la interfaz del dispositivo B, que soporta controlador en modo usuario para un escáner (*Usbscan.exe*) que se comunica con *Winusb.dll* a través de un servicio del sistema (*SVCHOST*).

- Caso 3 de *winusb.sys*: Registra la interfaz del dispositivo C, que soporta una utilidad de actualización del firmware (*Usbfw.exe*).

Hay exactamente una instancia cargada de *winusb.sys* para cada objeto de dispositivo físico (ODF). Un ODF puede representar un dispositivo compuesto (por ejemplo, en el caso 1 de la figura C.1) o puede representar una interfaz o una colección de interfaces de un dispositivo compuesto (por ejemplo, los casos 2 y 3).

Cualquier aplicación en modo de usuario puede comunicarse con la pila USB cargando la biblioteca de enlaces dinámicos de WinUsb (*Winusb.dll*).

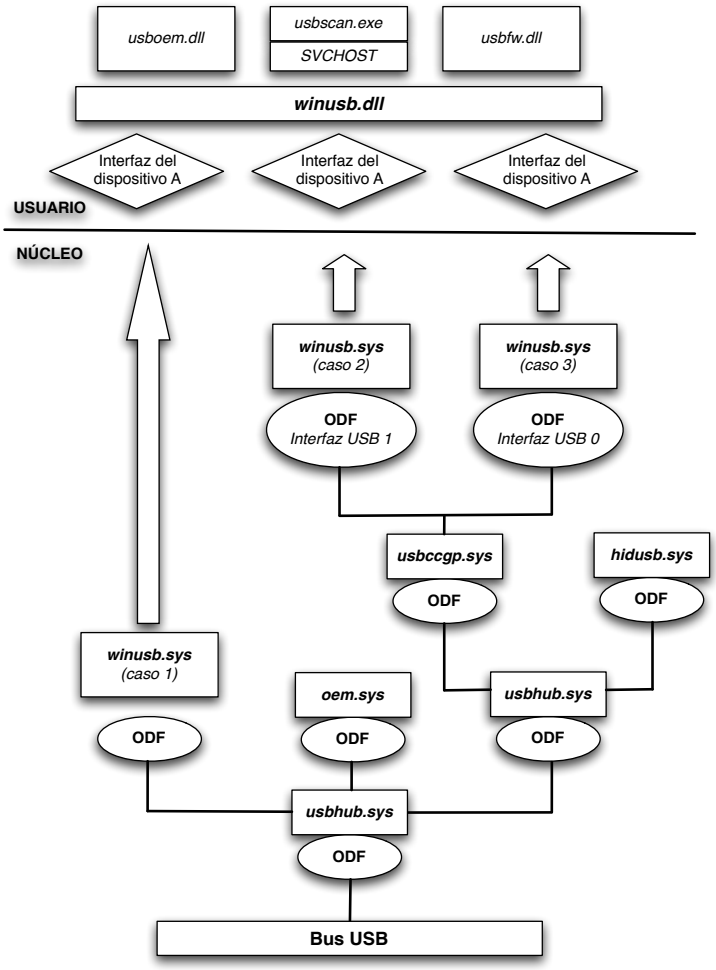

Figura C.1: Diagrama de la arquitectura de WinUsb

C.2 Instalación de *Winusb.sys* como un controlador de función

Antes que una aplicación pueda hacer uso de la API de WinUsb para comunicarse con un dispositivo, se necesita instalar *Winusb.sys* como el controlador de función del dispositivo.

Tanto la API de WinUsb y como el instalador para *Winusb.sys* se encuentran en el *WDK (Windows Driver Kit)* proporcionado por Microsoft.

Para su instalación, es necesario un crear un paquete que incluye lo siguiente:

- El archivo *WinUSB co-installer*, el cual instala instala WinUSB en el sistema ope-

rativo, si es necesario. El *WDK* incluye dos versiones de *co-installer*, uno para los sistemas x86 y uno para los sistemas x64. Ambos tienen el nombre *WinUSBCo-Installer.dll* y están localizados en la carpeta *WinDDK\BuilNumber\redist\winusb*.

- El *KMDF co-installer*, se encarga de instalar la versión correcta de KMDF en el sistema operativo, si es necesario. Este *con-installer* es necesario por que *Winusb.sys* depende de KMDF. El nombre del *co-installer* van en función a la versión de KMDF, en este caso su nombre es *WdfCoInstaller01005.dll* por que está asociada al la versión KMDF 1.5. Las versiones x86 y x64 de *WdfCoInstaller.dll* se incluyen en el *WDK* en la carpeta *WinDDK\BuilNumber\redist\wdf*.

- Se necesita crear un archivo INF que instala *Winusb.sys* como el controlador de función del dispositivo.

- Un archivo de catálogo firmado por el paquete. Este archivo solo es necesario para instalar WinUSB en las versiones x64 de Windows Vista.

Después de crear el archivo INF, se instala *Winusb.sys* en la misma forma cualquier otro controlador. El método más simples es conectar el dispositivo y utilizar el asistente para agregar un nuevo hardware o el administrador de dispositivos para instalar el controlador utilizando el INF.

Bibliografía

[1] Flores Ortega Juan Pablo. Tesis: *Interfaz avanzada de tiempo real para la medición multidimensional del estrés*. Cinvestav 2006.

[2] Bonilla Henríquez Enrique. Tesis: *Medidor multidimensional de estrés*. Cinvestav 2005.

[3] Pérez Olán Gregorio. Tesis: *Medidor multidimensional de estrés*. Cinvestav 2004.

[4] De Luca Pennacchia Adriano, Acevedo M. María., *Sistema de Medición de Estrés*, Revista Mexicana de Ingeniería Biomédica, vol. XXII, núm. 1, Enero-Marzo 2001, pp. 20-25.

[5] Di Nuovo S., Rispoli L., Genita E., *Misurare lo Stress. Il test MSP e altri strument per una valutazione integrata*, Ed. Franco Angeli/Linea Test año 2000.

[6] Hans S., *The Stress of Life*, Mc Graw-Hill 1956.

[7] Guerrero Gutiérrez, Laura E., García Sánchez, Maximino, *Indices de perfil estrés y enfermedades crónicas en pacientes del ISSSTE*, Universidad del Noreste, Área de ciencias del Comportamiento y Educación, México, 2007.

[8] Silpak, Oscar., *Concepto del estrés*, 1ra. parte, Revista Alcmeón, vol. 1, núm. 3, Buenos Aires 1991.

[9] Allen Jhon ,*Photoplethysmography and its application in clinical physiological measurement*, Regional Medical Physics Department, Freeman Hospital, Newcastle upon Tyne NE7 7DN, UK. Publicado: 20/ Febrero/ 2007.

[10] López Silva, S. M., Silveira, J. P., *et. al.*, *Utilidad de la fotopletismografía por transmisión con diodos láser infrarrojos en el estudio de la perfusión visceral: estudio preliminar*, Óptica Pura y Aplicada, vol. 38, núm. 1, 2005.

[11] Martínez González, Daniel, *Sistema autónomo para la medida óptica del ritmo cardíaco*, Universidad Politécnica de Cataluña, España 2008.

[12] Posada Gomez Rubén, Enriquez Rodíguez Jose J. *et al.*, *USB bulk transfers between a PC and PIC microcontroller for embedded applications*, Instituto Tecnológico de

Orizaba, División de estudios de investigación y postgrados. Electronics, Robotics and Automotive Mechanics Conference 2008.

[13] Sheng Liang, *The Java Native Interface: Programmer's Guide and Specification*, Sun Mycrosystems Inc., Ed. Addison Wesley, Junio 1999.

[14] Arenas Mas, Marta. *Diseño e implementación de un sistema de adquisición de aceleraciones con procesamiento mediante microcontrolador*. Departamento de Ingeniería Eléctrica, Universidad de Sevilla, España. Junio 2008.

[15] Ordaz Guzmán, Rolando. *Interfase USB/IDE*. Universidad de las Américas Puebla, México, Mayo 2004.

[16] Vila Sobrino, José A., *Análisis de la variabilidad de señales fisiológicas: Integración en un sistema de monitorización inteligente*. Universidad de Santiago de Compostela, Departamento de Electrónica e Computación. España, Abril 1997.

[17] García González, Miguel A., *Estudio de la variabilidad del ritmo cardíaco mediante técnicas estadísticas, espectrales y no lineales*. Universidad Politécnica de Cataluña, España, Octubre 1998.

[18] DeBenedittis G., Cigada M., *Autonomic balance during Hypnosis and Heart Rate Variability Power Spectrum analysis as a marker of sympatho-vagal balance*, International Journal of Clinical Experimental Hypnosis, vol. 42, pp.140-152, 1994.

[19] Maud PJ. , Foster C. *Physiological assessment of human fitness*, Second Edition. 2006.

[20] Appel, M.I., *et. al.*, *R.J.Beat to beat variability in cardiovascular variables: Noise or music?*, Harvard-MIT Division of Health Sciences and Technology, Massachusetts Institute of Technology, Cambridge 1989; 14, 5, pp.1139-1148.

[21] Malliani, A., Pagani, M., Lombardi, F., & Cerutti, S., *Cardiovascular neural regulation explored in the frequency domain*. Istituto Ricerche Cardiovascolari, Centro Ricerche Cardiovascolari, CNR, Milano, Italy. 1991. Circulation, vol. 84, pp.482-492.

[22] Akselrod, S., Gordon, D., Ubel F.A., *Power spectrum analysis of heart rate fluctuation: a quantitative probe of beat-to-beat cardiovascular control*. Science 1981, vol. 213, no. 4504, pp.220-222.

[23] Sandrine C. Millasseaua, *et. al.*, *Contour analysis of the photoplethysmographic pulse measured at the Þnger*, Journal of Hypertension 2006, vol 24, pp.1449-1456.

[24] M.K. Ali Hassan, *et. al.*, *Measuring Blood Pressure Using a Photoplethysmography Approach*, Mechatronic Engineering Programme, School of Mechatronic Engineering, University Malaysia Perlis, 2008.

[25] G. D, Jindal, *et. al.*, *Non-invasive Assessment of Blood Glucose by PhotoPlethysmography*, Electronics Division, BARC Mumbal, India. IETE Journal of Research, vol 54, no. 3, pp. 217-222, May-June 2008.

[26] D. G. Bruce, *et. al.*, *The effects of sympathetic nervous system activation and psychological stress on glucose metabolism and blood pressure in subjects with Type 2 (non-insulin-dependent) diabetes mellitus*, Garvan Institute of Medical Research, St Vincent's Hospital, Sydney, Australia. Diabetologia , vol 35, pp. 835-843, 1992.

Páginas Web

[27] Copernal Publishing, S.L., *Estrés, la respuesta de nuestro organismo* [en línea], 2009, [Citado: 2-Diciembre-2009], `http://www.estresansiedad.com/Estres-la-respuesta-de-nuestro-organismo/12`.

[28] Puras Mallagray, Enrique, *Pletismografía: presiones segmentarias*, Fundación Hospital de Alcorcón, Madrid [en línea], 2009, [Citado: 2-Diciembre-2009], `http://www.cdvni.org/pdf/PrincipiosV.pdf`.

[29] Microchip Technology Inc. *PIC18F2455/2550/4455/4550 Data Sheet: High-Performance, Enhanced Flash, USB Microcontrollers with nano Watt Technology* [en línea], 2009, [Citado: 2-Diciembre-2009] `http://www.microchip.com`.

[30] Brodín Trujillano, Eloy. Giménez Pastor, Adrián. *El bus USB (Universal Serial Bus)* [en línea], 2009, [Citado: 5-Diciembre-2009], `http://usuarios.multimania.es/kurganz/index.html`.

[31] Soporte Técnico OEM, Fujitsu España, *Un paseo por USB 2.0* [en línea], 2003, `http://www.fujitsu.com/downloads/EU/es/soporte/discosduros/UnpaseoporUSB-2.pdf`

[32] USB Implementers Forum, Inc. *Universal Serial Bus Revision 2.0 specification* [en línea], 2009, [Citado: 5-Diciembre-2009], `http://www.usb.org/developers/docs/`.

[33] Axelson, Jan. *Jan Axelson's Lakeview Research: WinUSB* [en línea], 2009, [Citado: 10-Febrero-2009], `http://www.lvr.com/winusb.htm`.

[34] Microsoft Corporation. *WinUSB* [en línea], 2009, [Citado: 15-Febrero-2009], `http://msdn.microsoft.com/en-us/library/aa476426.aspx`.

[35] Microsoft Corporation. *How to Use WinUSB to Communicate with a USB Device* [en línea], 2009, [Citado: 15-Febrero-2009], `http://www.microsoft.com/whdc/connect/usb/winusb_howto.mspx`.

[36] López Hernández Fernando. *JNI: Java Native Interface* [en línea] 2007, [Citado: 12 -Marzo-2009], http://macprogramadores.org/tutoriales/tutoriales/jni.pdf.

[37] Sun Mycrosystems Inc. *JNI 1.1 Specification* [en línea], 2001, [Citado: 12-Marzo-2009], http://java.sun.com/j2se/1.4.2/docs/guide/jni/spec/jniTOC.html.

[38] Microsoft Corporation, *About Dynamic-Link Libraries* [en línea], 2009, [Citado: 12-Marzo- 2009], http://msdn.microsoft.com/en-us/library/ms681914(VS.85).aspx.

[39] IEEE, *IEEE standard glossary of software engineering terminology 610.12-1990*, 1990,[Citado: 12-Marzo- 2009], http://ieeexplore.ieee.org/servlet/opac?punumber=2238.

[40] Custom Computer Services, Inc. *C Compiler: Reference Manual August 2009* [en línea], 2009, [Citado: 12-Marzo- 2009] , http://www.ccsinfo.com/downloads/ccs_c_manual.pdf.

[41] Rojas Contreras, Paola Andrea. *Diseño de microcontroladores: Watchdos Timer*, Universidad Técnica Federico Santa María, Departamento de Eléctronica, [en línea], 2004, [Citado: 5-Diciembre-2009], http://www.elo.utfsm.cl/~lsb/elo325/clases/WATCHDOG%20TIMER.pdf.

[42] Hoffman, Pablo. Szmulewicz, Martín. *Osciloscopio USB*. Universidad ORT Uruguay, Facultad Ingeniería. Uruguay, [en línea], 2006, [Citado: 8-Diciembre-2009] http://pablohoffman.com/twiki/pub/Oscusb/OscusbDocumentacion/oscusb-documentacion.pdf.

[43] Beyond Logic Inc. *USB in a NutShell: Making sense of the USB standard* [en línea], 2009, [Citado: 15-Febrero-2009], http://www.beyondlogic.org/usbnutshell/usb5.htm

[44] Institute of HeartMath, *Science of The Heart: Exploring the Role of the Heart in Human Performance* 2009, [Citado: 10-Agosto-2009], http://www.heartmath.org/research/science-of-the-heart.html

[45] National Institutes of Health, NIBIB & NIGMS., *PhysioToolkit binaries for MS-DOS/MS-Windows* [en línea], 2009, [Citado: 15-Agosto-2009], http://www.physionet.org/physiotools/binaries/windows/

[46] Romero Méndez, Juan P., *Máxima Entropía* [en línea], 2008, [Citado: 16-Agosto-2009], http://www.fenomec.unam.mx/pablo/seminario/maxima_entropia.pdf

[47] Elemaya Inc., *Heart rate variability: cosa e' ed applicazione* [en línea], 2009, [Citado: 16-Agosto-2009], `http://www.elemaya.com/XHeartvar.htm`