Centro de investigación y de estudios avanzados
del Instituto Politécnico Nacional.
Unidad Zacatenco.

Departamento de Ingeniería Eléctrica
Área de Computación

Interfaz avanzada de tiempo real para la medición multidimensional del estrés.

Tesis que presenta el
Ing. Juan Pablo Flores Ortega

para obtener el grado de **Maestro en ciencias**
en la especialidad de **Ingeniería Eléctrica**

Director de Tesis

Dr Adriano de Luca Pennacchia.

Mexico D.F 6 de Noviembre del 2006

A mi esposa e hija por ser la causa, razón y motivo que me impulsan a tratar de ser mejor cada día, su amor, alegrías y logros me impulsaron a llegar a esta meta.

Es de ustedes.

Agradecimientos

A dios por que siempre ha estado presente en los momentos más importantes de mi vida, fuente de inspiración, conocimientos y guía sin condiciones ni restricciones.

Con mucho cariño a mis padres por haberme dado la vida, la educación, la determinación y la integridad de ser quien soy, nunca cambien.

Con especial amor a mi esposa Nora Alejandra y mi hija Nora Mariana que siempre me están apoyando para que juntos seamos una gran familia.

A todos los maestros que de alguna u otra forma nutrieron mi mente y espíritu con su saber y conocimientos.

A los verdaderos amigos que solo se cuentan con los dedos de las manos.

Un sincero agradecimiento a mi director de tesis Adriano de Luca Pennacchia.

Un agradecimiento a todo el personal administrativo del CINVESTAV, en especial a Sofía Reza.

Resumen

La medición del stress, desde su descubrimiento, ha sido uno de los retos de psicólogos y médicos en general. El determinar si una persona está afectada por el stress o no, permite definir el tipo de tratamiento a aplicar en un paciente. Los métodos para realizar este tipo de evaluaciones son mediante pruebas psicológicas y fisiológicas.

El trabajo que aquí se presenta es una combinación de ambos, tanto de pruebas psicológicas como de evaluaciones fisiológicas. A este sistema de medición del stress se le ha denominado Medidor Multidimensional de Stress.

En trabajos anteriores se ha mejorado el test que se encarga de dar un parámetro del estrés fisiológico, en un paciente, corresponde a este trabajo el mejorar el módulo encargado de la medición del estrés fisiológico. Dichas mejoras consisten en aumentar la cantidad de muestras obtenidas por segundo de un paciente, así como la correspondiente graficación y almacenamiento de los mismos.

Toda la información que el sistema genere debe ser guardada y posteriormente interpretada, por personal clínico, psicólogos y personas interesadas en el estudio del stress. La información de las pruebas con el medidor se guardaban inicialmente en archivos, lo que dificultó su manejo a gran escala. De ahí la necesidad de implementar un nuevo esquema de manejo de toda la información proveniente de la parte de hardware y software.

Esquema en el cual se usa programación en java para permitir portabilidad y fácil mantenimiento del programa y una base de datos que permite organizar la información de manera que permita un rápido acceso y análisis de la información proveniente del hardware.

Índice General.

Índice de Imágenes

1 Antecedentes, motivación, objetivos y metodología.

1.1 Antecedentes

El sistema descrito en los siguientes capítulos de este documento tiene como finalidad el presentar el diseño e implementación de la Interfaz Avanzada de Tiempo Real para la Medición Multidimensional del eStrés (IATRMMS). Trabajo basado en el sistema Computacional para la Medición Multidimencional del Estrés (SCMME) [28], el cual a su vez esta basado en el trabajo desarrollado por Gregorio Pérez Olán [21]. El cual permitía la evaluación fisiológica y psicológica del nivel de estrés crónico de las personas.

El estudio del efecto del estrés crónico en las personas llevó al Dr. Adriano de Luca a solicitar la construcción de dicho sistema que permitiera su medición. Esta construcción condujo a la implementación del sistema SCMME con buenos resultados. Sin embargo, el diseño monolítico de la aplicación no permitió realizar modificaciones y mejoras de manera fácil y confiable.

De ahí la necesidad de rediseñar el antiguo sistema para ofrecer un sistema modular que tuviera mejor definición estructural. Para esta reestructuración se determinó seccionar todo el sistema en módulos. Con la propuesta de este nuevo Sistema Modular para la Medición del Estrés se pretende tener una plataforma que permita el desarrollo de cambios rápidos y precisos en su diseño, en base a la modularidad (solo es necesario corregir los módulos implicados en los cambios). Esto fue necesario ya que el estudio sobre la medición del estrés es un campo amplio y con grandes posibilidades para la investigación, que requiere de un sistema modular que facilite esta investigación y acelere la implementación. Los nuevos sistemas se podrán implementar utilizando como base los módulos diseñados e implementados en este trabajo. Se pretende realizar también un aumento en la velocidad de muestreo de datos en la parte de evaluación fisiológica, gracias al uso del estándar USB 2.0
.

1.2 Motivación

Para darse una idea del funcionamiento e implementación de las primeras versiones del sistema, se mencionarán las características concretas de los diseños anteriores y algunos inconvenientes, propios de la tecnología usada.

Primer diseño

- Fue desarrollado por Gregorio Pérez Olán [21].
- Utiliza el estándar USB 1.0 como protocolo de comunicación con la PC.
- El microcontrolador utilizado es el 16C745.
- El *firmware* fue programado en lenguaje ensamblador.
- Utilizaron un circuito acelerómetro para medir la señal fisiológica abdominal.
- El *Software* (interfaz de usuario) fue programado en C++.

Deficiencias del primer diseño

- Existe pérdida de datos en la comunicación tarjeta - PC.
- Solamente se muestrea un dato cada 10ms.
- El diseño no es modular.
- No se calcula correctamente la distancia R-R [ver apéndice A].
- Utiliza costosas tarjetas de adquisición de señales fisiológicas.

Segundo diseño

- Fue desarrollado por Enrique Bonilla Henríquez [28].
- Utiliza el estándar USB 1.0 como protocolo de comunicación con la PC.
- Utiliza el mismo microcontrolador que la primera versión, el 16C745.
- El *firmware* también fue programado en lenguaje ensamblador.
- El *Software* (interfaz de usuario) fue programado en Java.
- El diseño del *software* de usuario fue separado en módulos (gráficador, procesamiento de los datos y administrador de base de datos).
- Se mejoro la velocidad en la transferencia de datos optimizando el código del *firmware*.

Deficiencias del segundo diseño

- Solamente el *software* de la interfaz de usuario se separó en módulos, el *firmware* solo se optimizo.
- Es necesario calcular la derivada de la señal electrocardiograma para calcular R-R, [ver apéndice A].
- Se sigue teniendo perdida de datos.
- Se utiliza un algoritmo de predicción del punto R, debido a la pérdida de datos.
- Existe la restricción de la velocidad (22ms) característica del protocolo USB 1.0 (10ms protocolo 12ms de datos)
- Utiliza costosas tarjetas de adquisición de señales fisiológicas.

En todos los diseños se tenía un gran inconveniente que era que se necesitaba "alambrar" al paciente, es decir, para conectarlo a cualquiera de los sistemas se le tenían que conectar por lo menos 3 cables a diferentes partes del cuerpo.

1.3 Objetivos

Analizando las deficiencias heredadas en los dos sistemas desarrollados anteriormente, así como los aciertos de ambos sistemas se decidió desarrollar una nueva versión del sistema intentando eliminar las deficiencias de los mismos.

Para el desarrollo de este nuevo sistema se plantearon los siguientes objetivos de manera que se cubrieran las deficiencias de los diseños anteriores:

1. Sustituir el uso de tarjetas de captura de señales fisiológicas para uso medico y desarrollar unas tarjetas propias de menor costo y con una resolución (número de tomas de datos) mayor a la ofrecida por las soluciones comerciales.

2. Utilizar el protocolo USB 2.0 el cual es más rápido que el anterior a fin de evitar la pérdida de datos.

3. Realizar el diseño del *firmware* (FW) de una manera modular para simplificar la actualización y reconfiguración del FW.

4. Utilizar *drivers* de fácil mantenimiento.

5. Hacer que el *software* de usuario sea amigable y de fácil instalación en cualquier computadora.

6. Buscar la manera de que la adquisición de señales fisiológicas sea de una manera no invasiva.

7. Mostrar con una gran resolución al usuario los valores adquiridos por el módulo de adquisición de señales fisiológicas

En base a esos objetivos fue que se de decidió dividir el proyecto en módulos independientes que al interconectarse cumplen con todos los objetivos listados anteriormente.

La descripción de cada uno de los módulos se mencionará a continuación, la descripción detallada del funcionamiento e implementación de cada módulo se explicará en los capítulos siguientes.

- Módulo de adquisición de señales fisiológicas en específico PPG.
- Módulo de transmisión basado en el protocolo USB 2.0.
- Módulo de interfaz de usuario programado en java.

Esta división de módulos se adopto por que simplifica el diseño de nuevos sistemas de captura de señales especificas, es decir, si quiero capturar otro tipo de señales fisiológicas como ECG [ver apéndice A] no es necesario que cambie el módulo de transmisión ni el de interfaz de usuario. Así mismo si cambio la interfaz de usuario por otra plataforma o su funcionamiento no tengo que hacer cambios en los otros módulos.

Como se mencionó anteriormente cada uno de estos módulos merece un capítulo específico que detalla su implementación y funcionamiento, más un módulo extra donde se integran sobre un circuito impreso los módulos de adquisición de señales fisiológicas y el módulo de transmisión usando el protocolo USB 2.0.

Cabe mencionar que actualmente en el mercado y en otras instituciones se están desarrollando dispositivos que cumplen solo con parte de los objetivos que se pretenden lograr en este trabajo y podrían considerarse como trabajo relacionado.
El primero de ellos es un anillo capaz de medir la señal PPG y enviarla a una computadora para su posterior análisis [11], dentro de la temática de envió de información de un dispositivo de captura a una computadora para su análisis también se han desarrollado opciones como la de utilizar bluetooth [30] para el envío de datos adquiridos por un dispositivo externo como un anillo. También existen trabajos en los que se capturan las señales fisiológicas utilizando dispositivos fotorreceptores que se colocan en el cuello del paciente [31] y trabajos relacionados en la captura de señales por dispositivos externos [19] y aplicaciones desarrolladas para uso comercial [33] pero ninguno de ellos implementa todas las características planteadas en los objetivos de este trabajo. También es cierto que existen varias maneras de determinar el nivel de estrés en un paciente pero hasta la fecha no se conoce algún trabajo que conjunte la medición multidimencional del estrés, es decir, solo se dedican a medir el estrés fisiológico o el psicológico, pero no conjuntan los resultados de ambas mediciones a fin de dar un tratamiento integral que permita la recuperación satisfactoria del paciente.

1.4 Metodología

En el desarrollo de los módulos específicos se tuvo que tomar en cuenta que cumplieran con los parámetros establecidos para la adquisición, muestreo y manipulación de los datos obtenidos de las mediciones del sistema.

En específico para el módulo de adquisición de señales fisiológicas se utilizó el esquema básico de adquisición y adecuación de señales fisiológicas el cual consiste en:

1. Utilizar un transductor para convertir los impulsos fisiológicos en una señal analógica, dicha señal analógica tendría un valor muy pequeño, del orden de nano Amperes.

2. Amplificar dicha señal analógica para poder trabajar con ella, es decir, una amplitud máxima de 5 Volts.

3. Filtrar la señal obtenida a fin de eliminar el ruido térmico y el producido por otros órganos y poder trabajar con una señal limpia.

4. Adecuar la amplitud de la señal a fin de hacerla compatible con el módulo siguiente

Para el módulo de transmisión basado en el protocolo USB 2.0, primero es necesario convertir la señal analógica proveniente del primer módulo a datos digitales para poder ser enviado a la PC. Una vez obtenidos estos datos digitales se hizo necesario utilizar un esquema donde se utilizara una memoria de doble puerto que almacena la señal fisiológica digitalizada y a su vez, es utilizada por el bloque de transmisión USB para leer los datos a ser enviados a la PC, el protocolo USB 2.0 no funciona por interrupciones [ver apéndice C] dicho protocolo especifica que es la PC la encargada de recibir los datos del dispositivo USB y no es el dispositivo USB el encargado de habilitar una interrupción de datos nuevos para que la PC reciba dichos datos. El esquema utilizado puede verse en la figura 1.

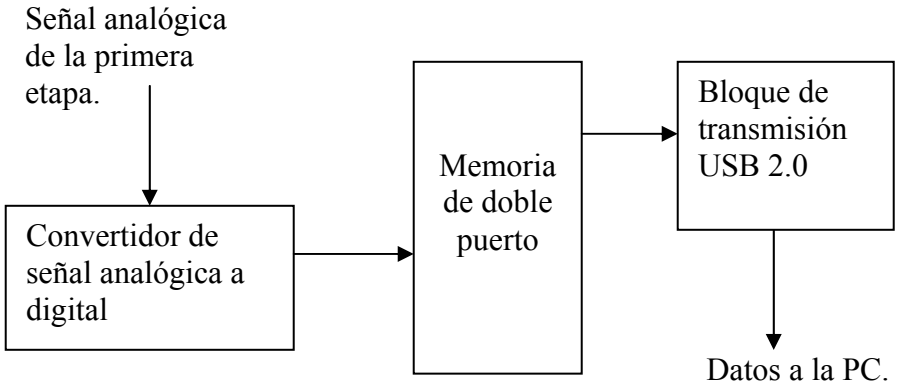


Figura 1 Esquema del sistema

En el desarrollo de este módulo también se intentó que los *drivers* utilizados fueran de fácil mantenimiento y compatibles con el sistema operativo.

Para el desarrollo del módulo de interfaz de usuario se decidió que se programaría en java para su desarrollo y consistiría de cuatro módulos básicos, los cuales son:

- Bloque de lectura de datos USB.
- Bloque de manejo de pacientes.
- Bloque de graficación de datos.
- Bloque de cálculo de R-R.

Cada uno de estos bloques se explicará a detalle en el capítulo cuatro.

También se dedica un capítulo a la integración de cada uno de los bloques explicados con detalle en un solo circuito impreso que cumple con todas las características deseadas en el planteamiento inicial del sistema.

Es el capítulo final donde se explican los logros alcanzados con este sistema así como sus características específicas y limitantes.

Además en los apéndices se incluye una pequeña explicación de las señales fisiológicas y su forma, una introducción a los amplificadores operacionales utilizados en el desarrollo del filtro y amplificación de la señal, una explicación de la especificación del protocolo USB 2.0 y los manuales de usuario, instalación del software de usuario, mantenimiento de FW y programación del Micro Controlador (MC).

2 Sensor de PPG (fotopletismografía o photoplethysmography)

La *pletismografía* incluye aquellas técnicas que miden cambios de volumen como consecuencia de variaciones del flujo sanguíneo. No son métodos específicos de un solo vaso arterial sino que miden cambios de volumen en un segmento de la extremidad. A continuación se mencionan algunas de ellas. Para pasar a la técnica que fue usada y finalmente la explicación de cómo se implemento a nivel *hardware*.

2.1 Tipos de pletismografía

- **Pneumopletismografía o Pletismografía de volumen de pulso.**

***Principios físicos*:** Consiste en la colocación de bandas de presión 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 mm Hg dependiendo de la localización de la banda de presión (muslo o dedos). Durante la sístole arterial se produce un incremento en el volumen de la extremidad que transmite presión contra la banda de presión rellena de aire, y a través de un sistema transductor de presión, ésta se convierte en una onda analógica de presión.

- **Pletismografía por Anillos de Mercurio.**

***Técnica*:** Consiste en la colocación de tubos de silicón rellenos de mercurio alrededor de la extremidad. La longitud de estos tubos con mercurio, será de 1 a 3 cm menor que la circunferencia de la extremidad. En los dedos esta longitud será de 0,5 cm menor.

***Principios Físicos*:** Con las expansiones y contracciones de la extremidad, la longitud del tubo de mercurio cambia. 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. Las características de las curvas obtenidas con pletismografía de anillos de mercurio son similares a las ya descritas para la pletismografía de volumen.

- **Fotopletismografía**

***Principios Físicos*:** Detecta el flujo de sangre cutáneo y traduce sus pulsaciones. Consiste en la emisión de luz infrarroja desde un diodo emisor y de un foto detector adyacente que recibe la luz infrarroja reflejada. A medida que aumenta el flujo de sangre cutáneo aumenta la cantidad de luz reflejada, ver figura 1. De esta manera obtenemos una medida cualitativa del flujo sanguíneo cutáneo. Se utiliza preferentemente en la medición de la presión digital.

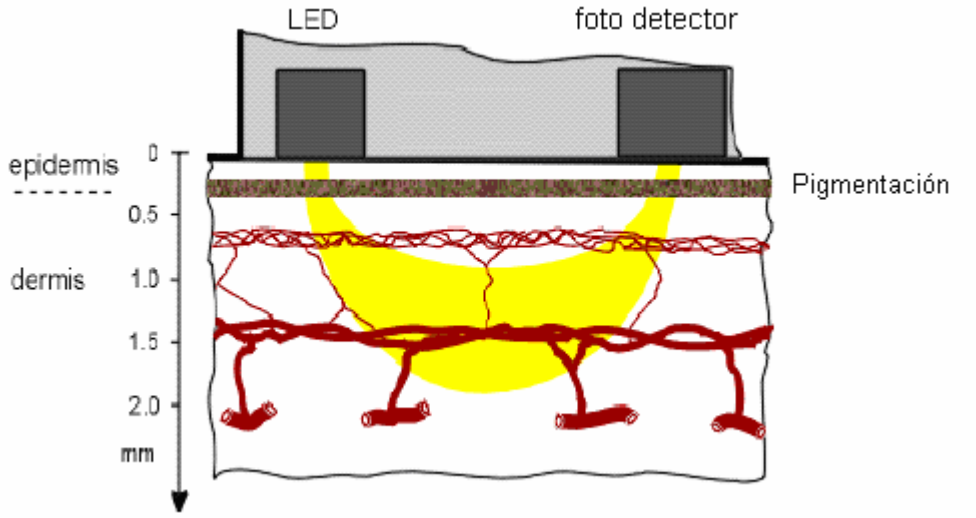


Figura 2 Principio fotopletismografía.

La micro-circulación de la sangre superficial es un asunto de gran interés fisiológico considerable debido a su papel en el metabolismo humano y la termorregulación.

Existen varias técnicas para analizar el flujo de la sangre sobre tejidos vivos, pero los métodos ópticos parecen ser los que mas prometen, puesto que la profundidad de penetración de la radiación óptica es relativamente pequeña comúnmente entre 0.1 y 3 milímetros dependiendo de la longitud de onda de la radiación. Por consiguiente 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 arterias y venas más profundas.

La fotopletismografía esta basada en la determinación de las propiedades ópticas de una determinada área de piel. Para este propósito luz infrarroja es emitida sobre la piel. La luz es absorbida en mayor o menor cantidad por la piel dependiendo de la cantidad de sangre que esta circulando en ese momento. Por consiguiente, la cantidad de luz reflejada corresponde con la variación del volumen de la sangre. Los cambios en el volumen de la sangre pueden ser determinados mediante la medición de la luz reflejada y las propiedades de la piel y la sangre.

Este efecto se puede apreciar a simple vista, la piel con menor volumen de sangre se ve blanca, mientras que la piel con mayor volumen de sangre luce más obscura, independientemente de la pigmentación de cada grupo étnico.

En el desarrollo de este sistema se optó por utilizar la fotopletismografía tomada en los dedos de las manos por ser el método menos invasivo de toma de señales. Que se tratará en la siguiente sección.

2.2 Técnicas para captura de fotopletismografía

A continuación se explican tres de las capturas usadas para realizar la fotopletismografía.

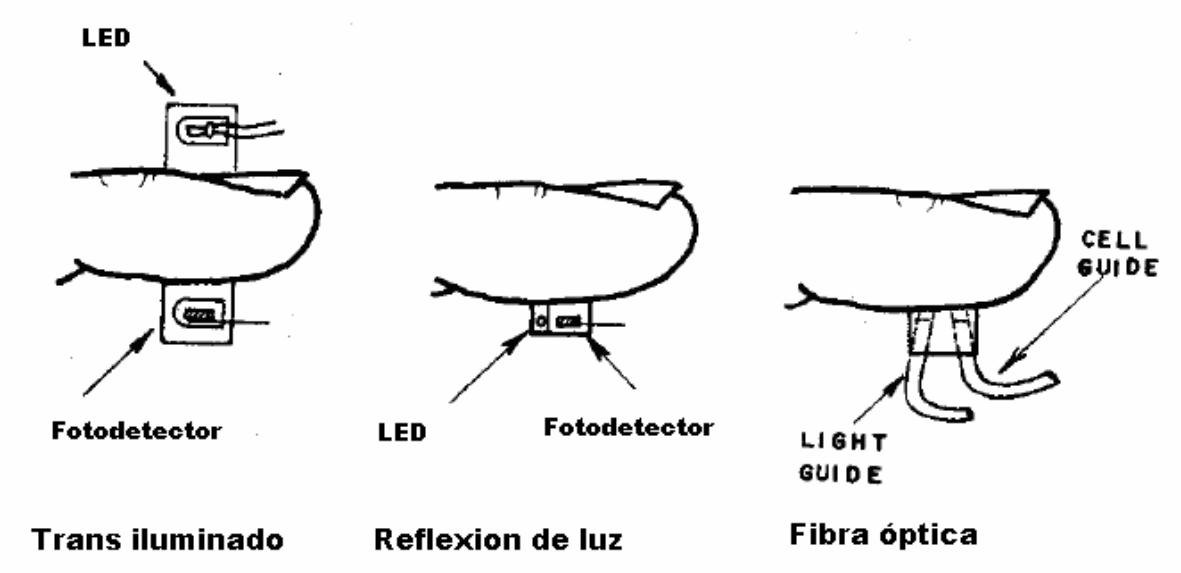


Figura 3 Tipos de captura fotopletismografía.

- **Trans-Iluminado.**

En esta técnica se colocan los dispositivos emisor y detector de tal forma que el dedo del paciente queda entre los mismos. Es necesario que en esta forma de tomar los datos la luz

infrarroja corresponda al espectro no visible para que la luz infrarroja pueda pasar a través del dedo.

- **Fibra Óptica.**

En esta técnica de captura de señal se utilizan 2 cables de fibra óptica en los cuales se hace circular 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.

- **Reflexión de luz (*backscatter*).**

Como su nombre lo indica en esta técnica de captura de señales superficiales se detecta la luz reflejada en mayor o menor cantidad por las venas superficiales esta técnica fue la que se decidió utilizar, por que se tienen las ventajas del tamaño reducido tanto del emisor como del receptor y el bajo costo de los mismos.

A continuación se explicará como se realizó la implementación en hardware del módulo de captura de PPG.

2.3 Implementación en hardware del módulo de captura PPG.

El módulo de captura de PPG de manera global consta de los siguientes bloques

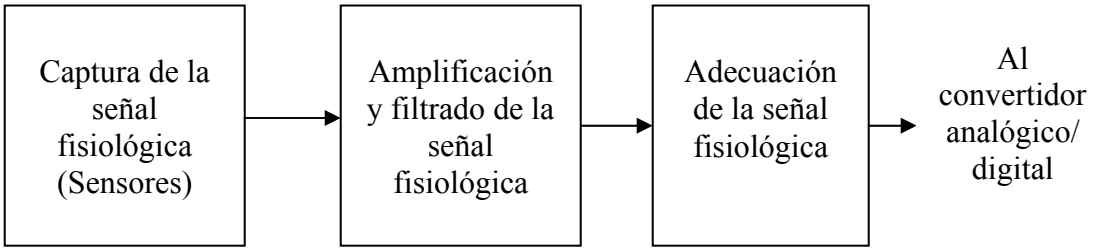


Figura 4 Diagrama de bloques

Una pequeña introducción a los amplificadores operacionales y sus distintas configuraciones que fueron usados para la realización de los distintos bloques de la figura 4, se encuentra en el apéndice B.

2.4 Captura de señal fisiológica

La sección de captura de señal fisiológica consta de un emisor de luz infrarroja (TLMK 3302) [32] y un diodo foto detector (BPW 34) [40].

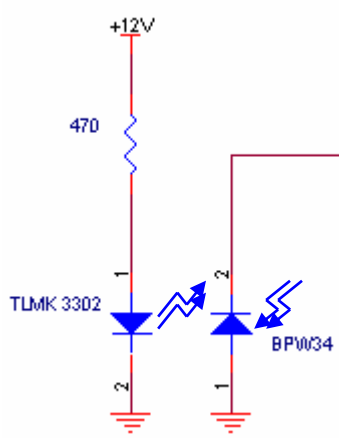


Figura 5 Captura de señales fisiológicas

El emisor de luz infrarroja (TLMK3302) esta polarizado a +12 V y tiene una resistencia de 470Ω para protegerlo contra la corriente. Las características de este emisor de luz infrarroja son las siguientes:

- Tiene una intensidad luminosa I_V=(400 a 800) mcd
- Emite luz infrarroja dentro del espectro visible.
- Tiene un tamaño reducido.

Parámetro	Condición de prueba	Símbolo	Min.	Tip.	Máx.	unidad
Intensidad luminosa	I_F = 50 mA.	I_V	400		800	mcd
Longitud de onda dominante	I_F = 50 mA.	λ_d	611	617	622	nm
Longitud de onda normal	I_F = 50 mA.	λ_p		624		nm
Ancho de banda espectral a 50% $I_{rel\ max}$	I_F = 50 mA.	$\Delta\lambda$		18		nm
Angulo de iluminación	I_F = 50 mA.	φ		±60		Grados
Voltaje directo	I_F = 50 mA.	V_F	1.85	2.1	2.55	V
Corriente inversa	V_R = 5 V.	V_R		0.01	10	µA

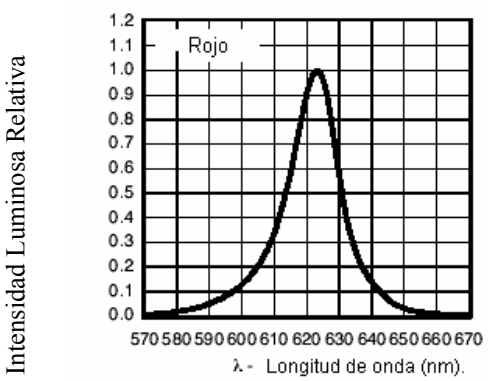


Figura 6 Intensidad relativa vs. longitud de onda.

Como el método de toma de datos usado es el de foto-reflexión es necesario que la luz infrarroja se encuentre dentro del espectro visible y este diodo emisor cumple con esa característica. Otra de las características por las que se escogió este dispositivo es su tamaño reducido.

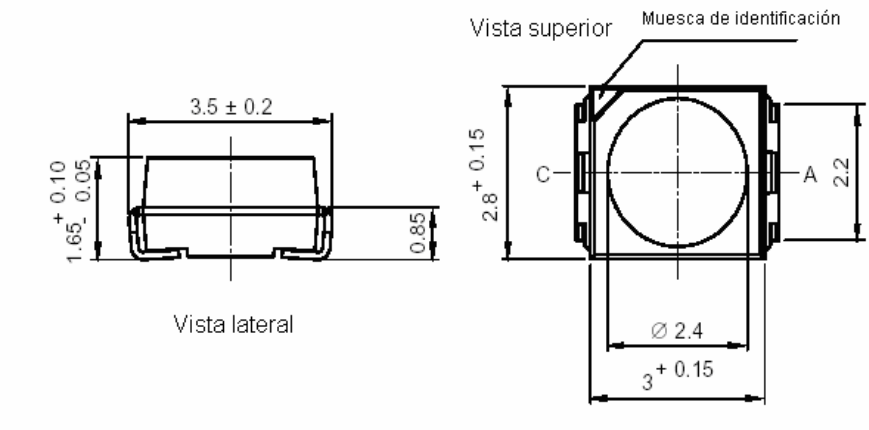


Figura 7 Dimensiones del emisor en milímetros.

Las dimensiones reducidas de este dispositivo conjuntamente con su bajo consumo de energía permiten que se pueda alimentar con la corriente suministrada por un puerto USB, lo que implica la reducción del circuito electrónico final.

El foto detector utilizado es el BPW34 [40] que tiene una superficie de detección de 7.5 mm^2, es altamente sensible, tiene un tiempo de respuesta muy alto, es sensible al espectro infrarrojo visible y una parte del espectro invisible. Veamos sus características.

Parámetro	Símbolo	Valor	Unidad
Voltaje inverso	V_R	60	V
Disipación de poder	P_V	215	mW
Temperatura de unión	T_j	100	ºC

Características Ópticas

Parámetro	Condición prueba	Símbolo	Típico	Unidad
Voltaje de circuito abierto	E_Θ = 1 mV/cm^2, λ=950 nm	V_O	350	mV
Coeficiente de temperatura de V_O	E_Θ = 1 mV/cm^2, λ=950 nm	TK_{Vo}	-2.6	mV/K
Corriente de circuito cerrado	EA = 1 klx	I_k	70	µA
	E_Θ = 1 mV/cm^2, λ=950 nm	I_k	47	µA
Coeficiente de temperatura de I_k	E_Θ = 1 mV/cm^2, λ=950 nm	TK_{Ik}	0.1	%K
Corriente de luz inversa	EA = 1 klx, V_R = 5 V	I_{ra}	75	µA
	E_Θ=1 mV/cm^2, λ=950 nm, V_R=5V	I_{ra}	50	µA
Angulo de sensibilidad		φ	±65	Grados
Sensible a la longitud de onda		λ_p	900	nm
Ancho de banda espectral		$\lambda_{0.5}$	600 a 1050	nm
Nivel de ruido	V_R = 10 V, λ = 950 nm	NEP	4 x 10 $^{-14}$	W/Hz
Tiempo de subida	VR = 10 V, RL=1 KΩ, λ 820 nm	t_r	100	ns
Tiempo de bajada	VR = 10 V, RL=1 KΩ, λ 820 nm	t_f	100	ns

Al igual que el diodo emisor de luz infrarroja este detector tiene un tamaño reducido lo que permite la portabilidad del par emisor receptor en un empaque pequeño.

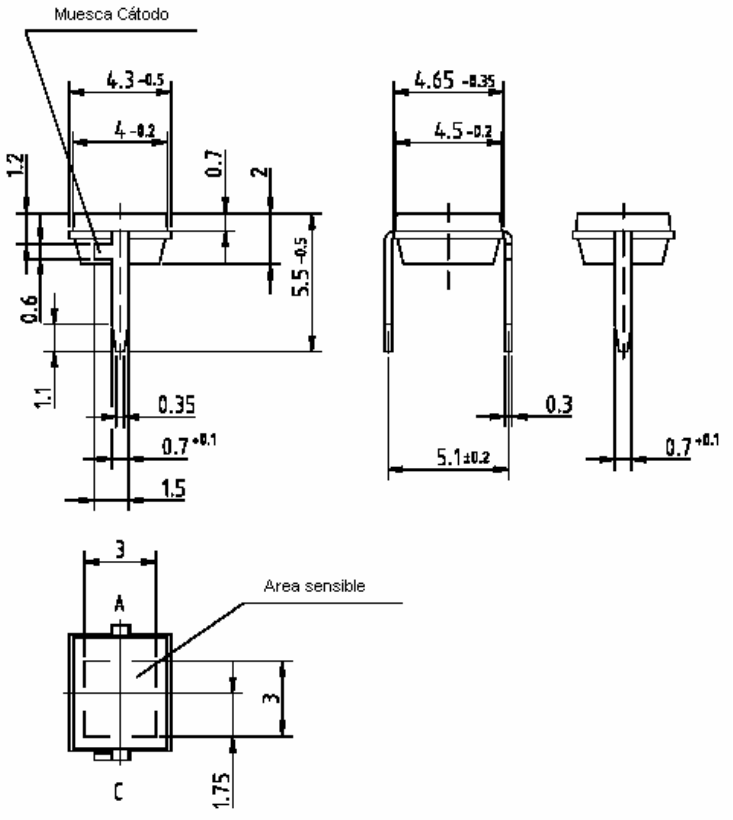


Figura 8 Dimensiones del receptor en milímetros.

Ambos dispositivos el emisor y el receptor fueron montados en una placa independiente del circuito, esto con el fin de poder colocar el par emisor-receptor en el dedo de la persona que se pretende hacer la captura de señal fisiológica. Quedando de la siguiente manera.

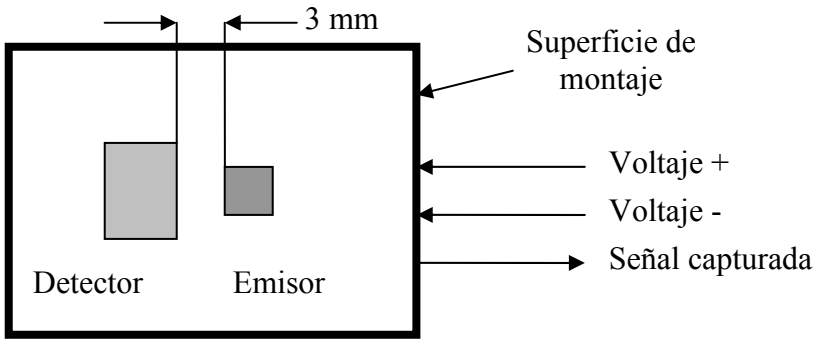


Figura 9 Montaje del emisor y el receptor.

Después de varias pruebas se llegó a la conclusión de que la distancia óptima entre emisor y detector se obtenía al colocarlos con una separación de 3 milímetros, pues es a esa distancia que se obtiene la señal con mayor amplitud y una menor cantidad de ruido.

El montaje del emisor y el receptor se hizo sobre una placa de circuito impreso normal, placa de cobre y base de baquetita, dejando 3 terminales, como se mostró en la figura 9, las cuales corresponden a la alimentación y la salida de la señal capturada, además se le agrego una cinta de velcro como se muestra en la figura 10.

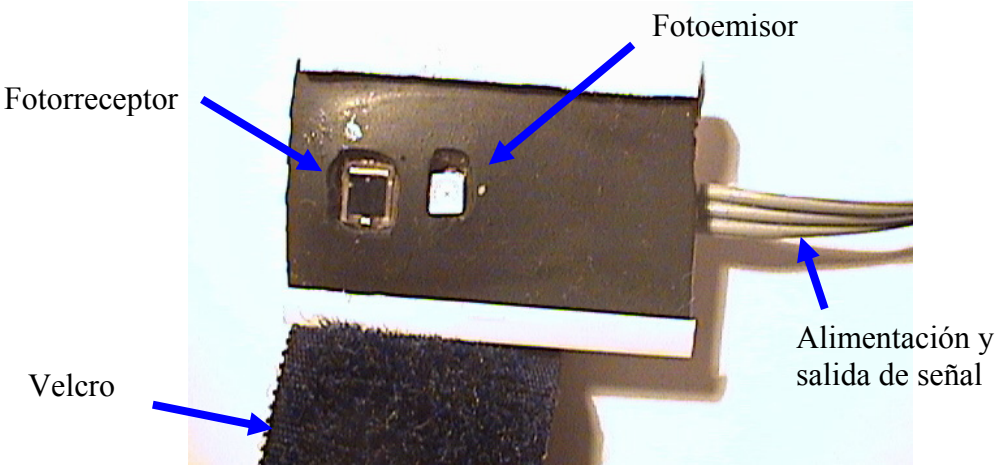


Figura 10 Placa receptora

Este tipo de captura llamada trans-iluminada ya se explico en la sección 2.2, la colocación de esta placa de captura es sobre los dedos de la mano como se muestra en la figura 11.

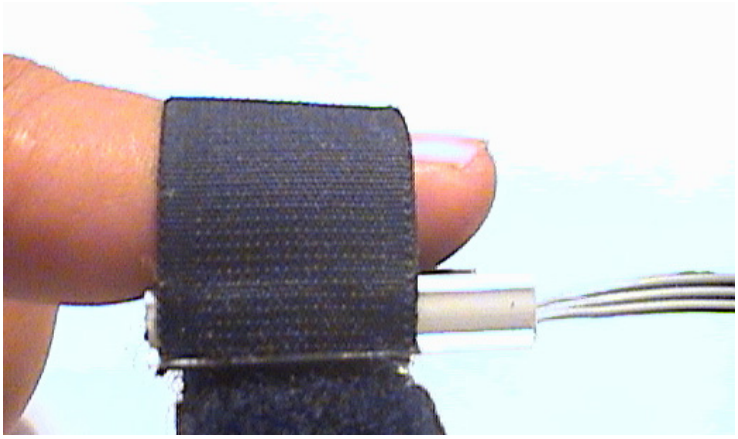


Figura 11 Dispositivo colocado en el paciente.

Los cables que sobresalen a la placa de captura se conectan al modulo de amplificación de señales.

Una vez que se ha explicado las características de los dispositivos de emisión y captura de señales infrarroja, se explicara el módulo de amplificación de la señal fisiológica.

2.5 Sección de amplificación y filtrado de la señal fisiológica.

Para la sección de amplificación de la señal capturada por el primer módulo se utilizó el circuito integrado TL084 [27]. El cual consta de 4 amplificadores operacionales, de bajo consumo de poder, con circuito de protección a la salida de los amplificadores, alta impedancia de entrada y compensación de frecuencia interna.

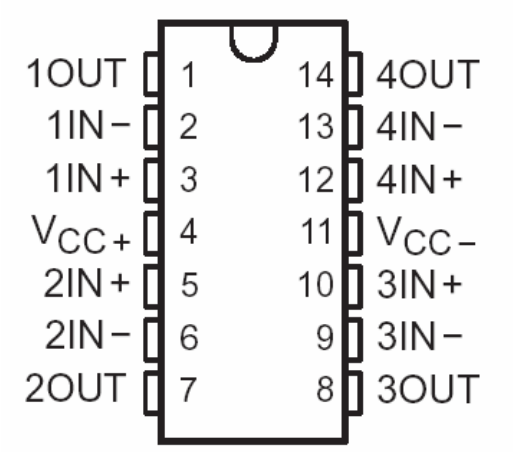


Figura 12 Amplificador operacional TL084.

Dicho circuito tiene la siguiente tabla de características eléctricas.

Parámetro	Símbolo	Valor	Unidad
Voltaje de alimentación	Vcc	±18	V
Voltaje de entrada	Vi	±15	V
Voltaje diferencial	Vid	±30	V
Disipación de potencia		680	mW

Tomando como base este circuito integrado se obtuvo el siguiente circuito resultante

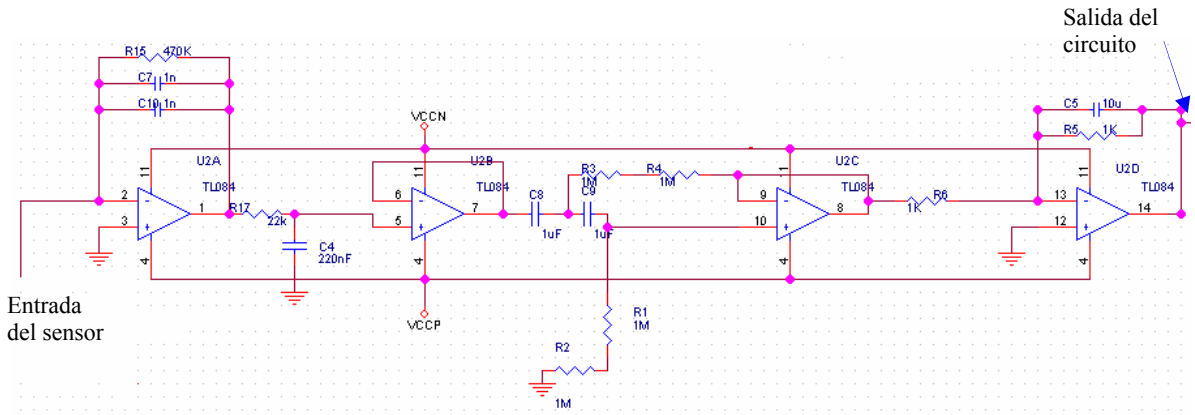


Figura 13 Configuración del circuito resultante.

Se puede observar que se utilizaron todos los amplificadores operacionales del circuito integrado cada uno de ellos funcionando en distintas configuraciones, las cuales se explican a continuación.

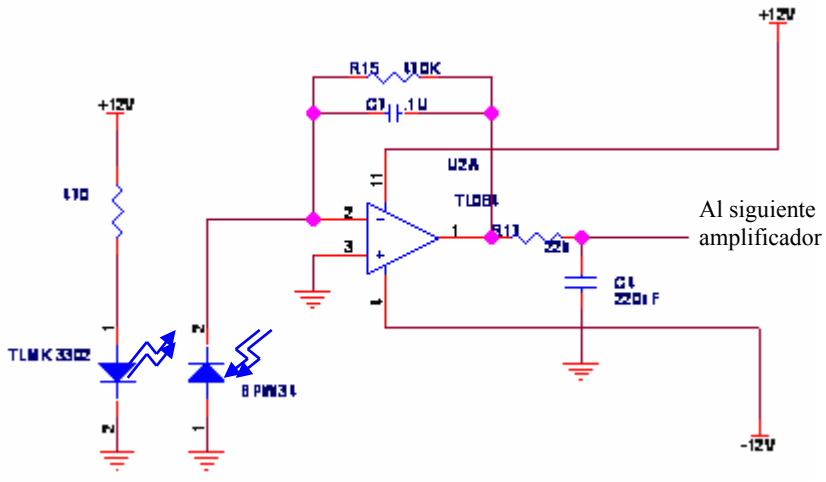


Figura 14 Primer amplificador operacional.

Se observa que el foto receptor BPW34 esta polarizado inversamente y su corriente de fuga depende del grado de luminosidad que reciba por efecto de la reflexión de luz infrarroja.

Experimentalmente se observó que la variación de corriente esta comprendida entre 30 y 60 nA en relación con la luz reflejada hacia el detector (valores máximo y mínimo). Este valor multiplicado por 470K, que es la amplificación del operacional, da un resultado de 15 a 30 mV a la salida del primer amplificador operacional.

La configuración del primer amplificador operacional es un pasa bajos de 10 Hz de segundo orden en donde el primer orden es dado por la combinación de la resistencia de 470KΩ con el capacitor de 0.1µF y el segundo orden esta dado por la resistencia de salida de 22KΩ y por el capacitor de 220nF.

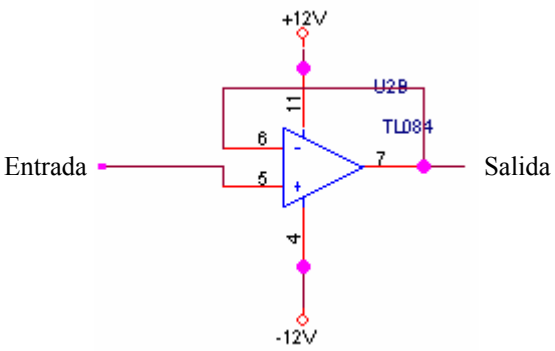


Figura 15 Segundo amplificador operacional.

El segundo amplificador operacional, mostrado en la figura 15, es usado solamente para desacoplar la parte de captura, amplificación y filtrado anterior, con el filtro pasa alto mostrado a continuación.

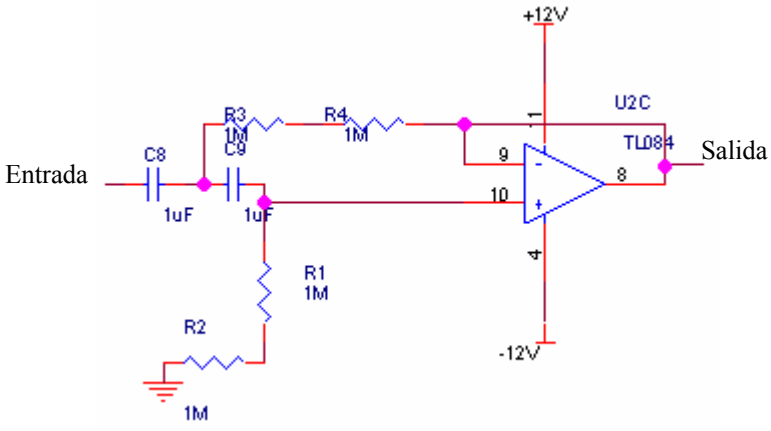


Figura 16 Tercer amplificador operacional.

La configuración de este tercer amplificador operacional da como resultado un filtro pasa altas de primer orden con una frecuencia de corte de 0.05Hz, es decir, deja pasar una frecuencia desde 0.05Hz en adelante. La razón de que se tengan 2 resistencias en serie de 1MΩ es que una de ellas es usada para calibrar la respuesta del filtro, puesto que no todos los amplificadores operacionales dentro un circuito integrado tienen una respuesta idéntica, esta resistencia puede ser sustituida por resistencias de precisión a fin de obtener la frecuencia de corte deseada. Esta frecuencia de corte es necesaria ya que permite eliminar las componentes de ruido no deseado dentro de la señal fisiológica a analizar como son ruidos térmicos (alimentación del circuito, calentamiento de los componentes) y ruidos producidos internamente por el cuerpo.

El último amplificador operacional dentro del circuito integrado es utilizado como amplificador de señal, y es mostrado en la figura 17.

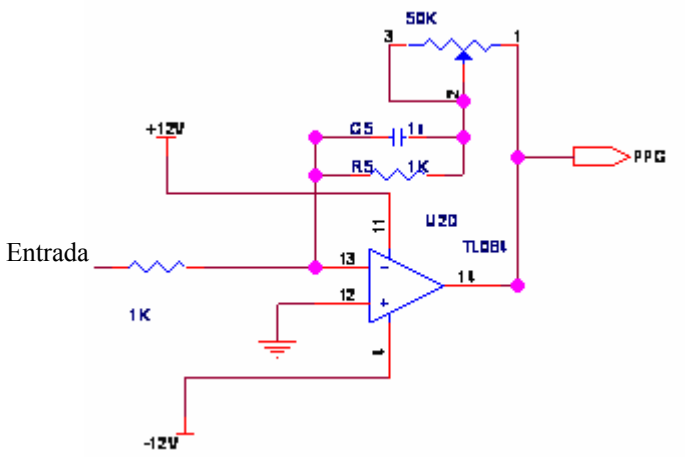


Figura 17 Cuarto amplificador operacional.

Esta configuración del amplificador operacional permite amplificar hasta 50 veces la señal de entrada, además es un filtro pasa bajo compuesto por la resistencia de 1KΩ en paralelo con el capacitor de 1μF. Lo que da un corte a 10Hz. En total en toda la línea tenemos.

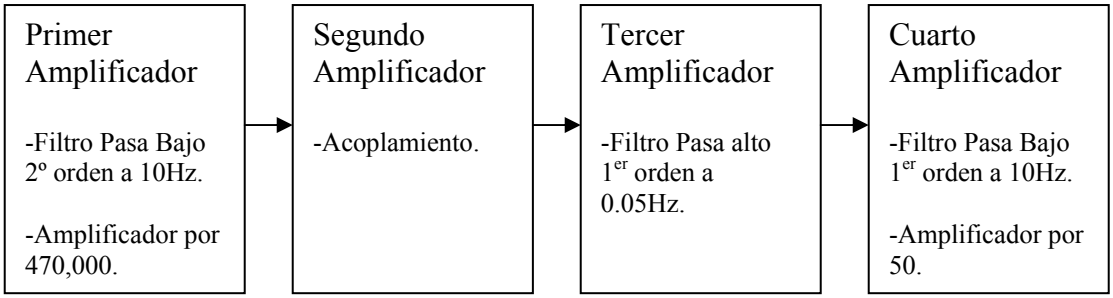


Figura 18 Etapas del proyecto.

Si unimos todas las etapas tenemos como resultado un filtro pasa banda con amplificación de señal, las frecuencias de la banda serian de 0.05Hz a 10Hz, la respuesta en frecuencia del circuito es mostrada en la figura 19.

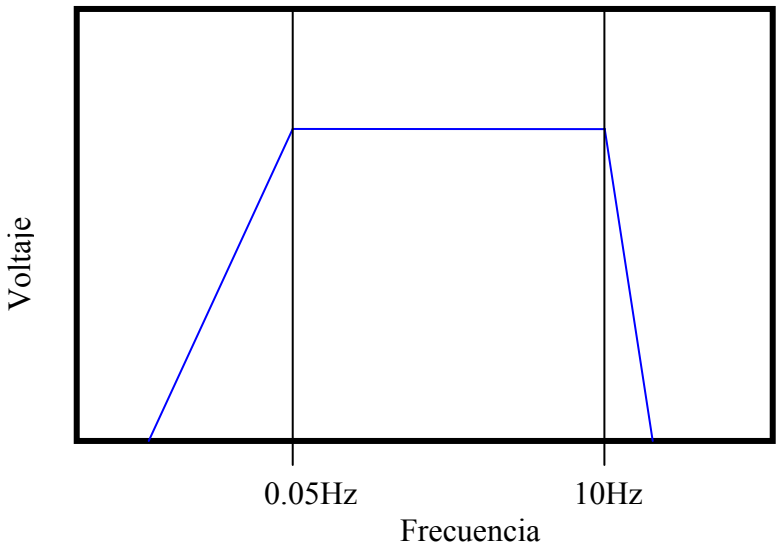


Figura 19 Respuesta en frecuencia del circuito resultante.

La pendiente en los flancos de la banda es diferente debido a que los filtros tienen diferente orden, el filtro pasa bajo es de orden 3 y el pasa alto orden 2. Mientras mayor sea el orden de un filtro más abrupta será la pendiente de la gráfica.

2.6 Sección de adecuación de señal fisiológica.

Esta parte esta conformada por un circuito que le agrega un nivel de *offset* a la señal de entrada, un inversor de señal y un circuito de desacoplo, para la implementación se utilizó un circuito integrado LM224N [41] el cual tiene las siguientes características.

- Elimina la necesidad de fuentes de alimentación duales.
- Los amplificadores operacionales están compensados en frecuencia internamente.
- Bajo consumo de energía, se puede alimentar con pilas.
- Ganancia de voltaje superior a 100 dB.
- Ancho de banda 1 MHz.
- Rango de alimentación
    - Fuente sencilla 3 a 32 Volts.
    - Fuente dual ±1.5 a ±16 Volts.
- Baja corriente de drenado de la fuente (700 μA).
- Baja corriente de entrada 45nA.

- Bajo nivel de voltaje de entrada 2mV con una corriente de 5nA.
- El rango de voltaje de entrada en modo común incluye tierra.
- El rango de voltaje de entrada diferencial puede ser igual al de la fuente de alimentación.

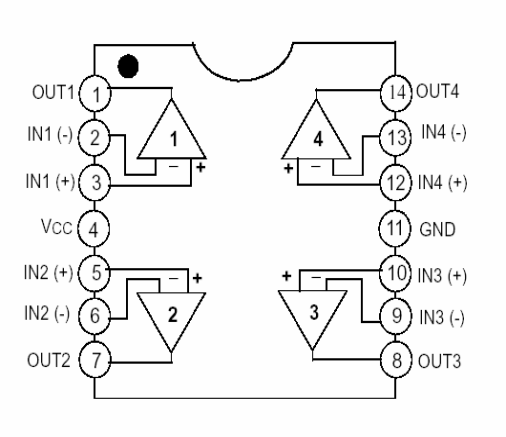


Figura 20 Amplificador operacional LM224.

Se utilizaron los cuatro amplificadores operacionales del circuito integrado resultando el circuito de la figura 21.

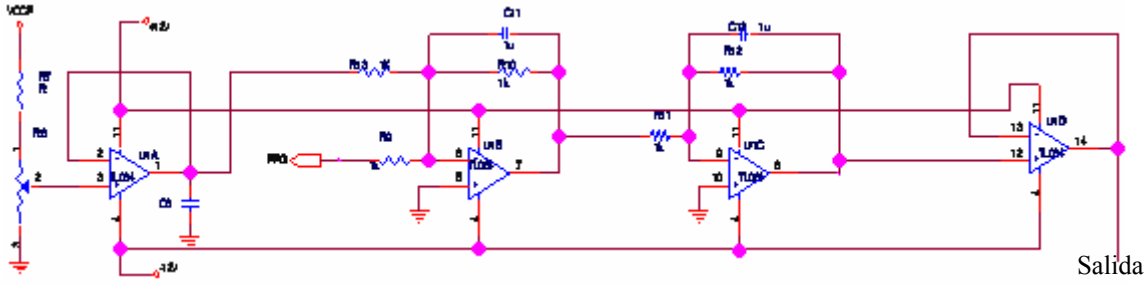


Figura 21 Configuración del LM224.

A continuación se explica como funciona cada parte del circuito.

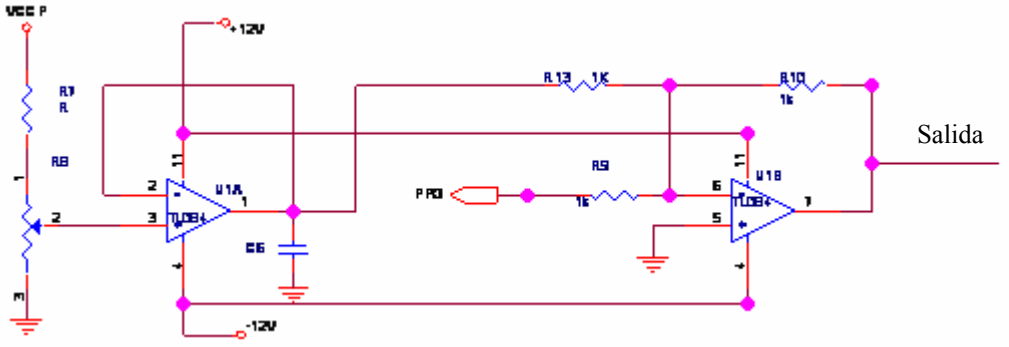


Figura 22 Inyector de nivel de *offset*.

Con estos dos amplificadores lo que hacemos es trasladar la señal que se encuentra oscilando entre valores positivos y negativos de la etapa anterior, a un nivel positivo de 0 a 5 Volts para poder entrar con valores positivos al bloque de conversión analógico digital del microcontrolador. El potenciómetro sirve para fijar el nivel de salida a 2.5 Volts. Con esto aseguramos que al momento de hacer la conversión analógico digital la señal no se pierda, pues el rango de entrada del convertidor analógico digital se encuentra entre 0 y 5 Volts. Se puede observar una representación grafica en la figura 23.

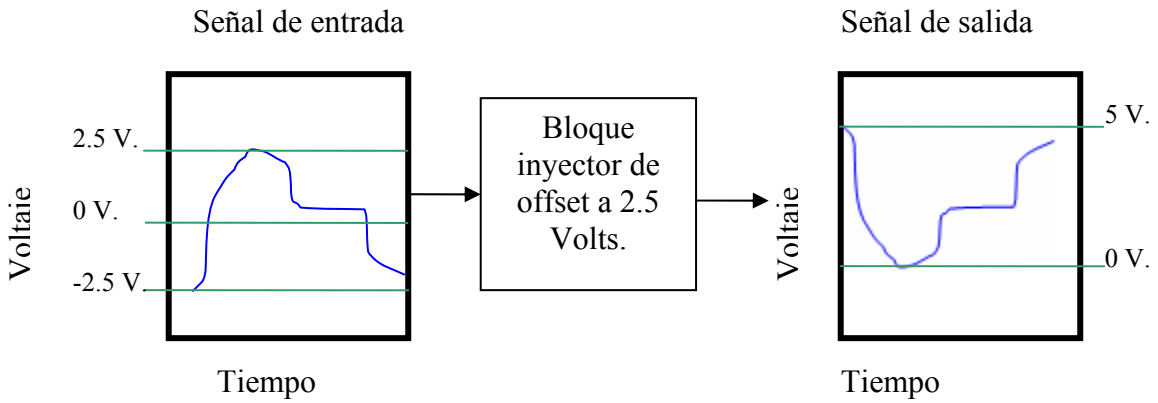


Figura 23 Esquema Funcional del Bloque.

El tercer y cuarto amplificador operacional son mostrados en la figura 24, con el tercer amplificador operacional se reinvierte la señal, pues el bloque inyector de *offset* invierte la señal original. Finalmente el cuarto amplificador operacional se utiliza para desacoplar el circuito anterior con la entrada del microcontrolador encargado de la conversión analógica digital.

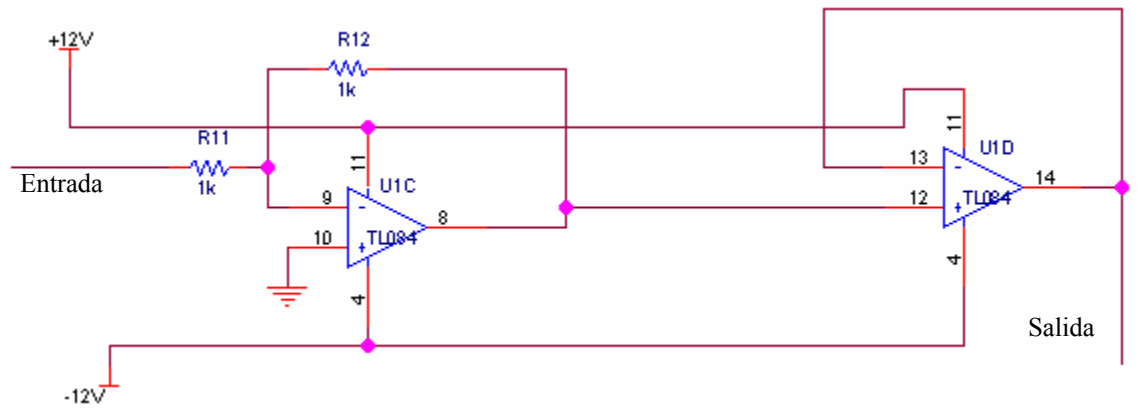


Figura 24 Bloque inversor y de desacoplo.

Con lo explicado anteriormente tenemos que el funcionamiento global en toda la línea para este circuito integrado es, un inyector de *offset* no inversor de la señal con desacoplo. En la figura 25 se puede observar el funcionamiento del bloque de una forma grafica.

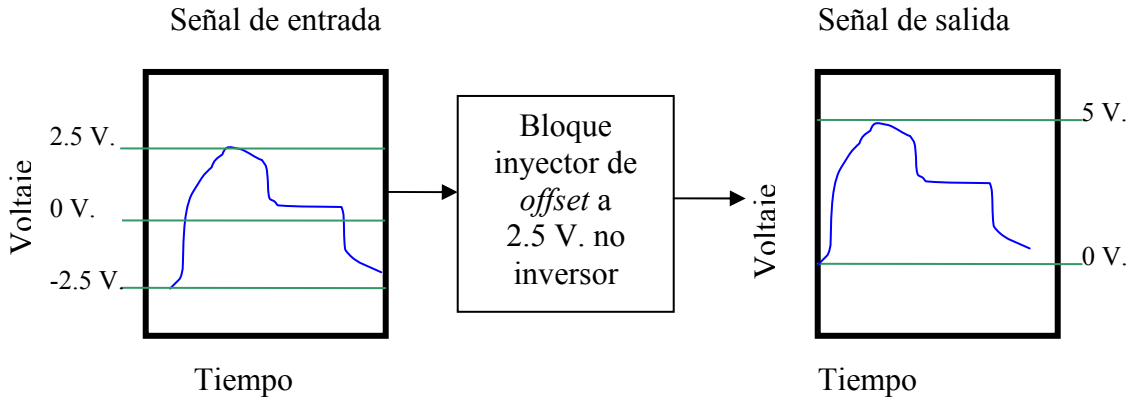


Figura 25 Esquema funcional del bloque.

Con estas características vemos que se cumple con los requisitos planteados en el primer capítulo, además de preparar la salida de este bloque con la entrada del siguiente que es la conversión analogía digital y la transmisión de estos datos usando el protocolo USB 2.0

Para implementar un sistema que se especialice en otra señal fisiológica como por ejemplo electrocardiograma (ECG), los únicos cambios que se tienen que realizar son cambiar los valores del filtro pasa bandas, explicado en la primer parte de este capítulo en la sección 2.5. Lo que implica solamente cambiar valores de capacitores y resistencias, pero no la configuración de los amplificadores operacionales. Esto nos permite en un mismo circuito impreso poder implementar distintos tipos de detectores de señales fisiológicas.

3 Transmisor USB 2.0

En este capítulo se hablara acerca de la implementación del módulo de transmisión de datos utilizando el protocolo USB 2.0, una pequeña introducción al protocolo USB 2.0 ha sido incluida en el apéndice C para su consulta, también se mencionaran los requerimientos de *hardware* y se terminara con la explicación de los principales módulos de programación del dispositivo, todas las características y configuraciones se pueden revisar mas a fondo en el *datasheet* del microcontrolador [22].

3.1 El protocolo USB

Como es sabido existen varias diferencias entre la versión 1.0 y 2.0 del protocolo USB (Ver apéndice C) dentro de las más importantes está la que permite enviar datos a una mayor velocidad y la mejora del sistema de recuperación de errores, estas fueron las principales características por las que se decidió utilizar dicho protocolo.
Como es sabido existen actualmente varias maneras de implementar el protocolo USB 2.0 a nivel *hardware* entre las que se encuentran varios microcontroladores (MC) los cuales difieren en tamaño y características de programación, tomando en cuenta que anteriormente ya se había trabajado con los MC ofrecidos por Microchip® se decidió utilizar los MC de la familia 18FXXXX. El MC seleccionado dentro de esta familia fue el 18F4550 que tiene las siguientes características:

- Tecnología nanoWat® (permite un consumo muy bajo de corriente).
- Memoria de programa Flash.
- Cumple con las especificaciones de USB 2.0.
- Permite una velocidad total “Full-speed” (12Mbit/s) y una velocidad baja “Low-speed” (1.5Mbits/s).
- Soporta transferencias por control, interrupción, *bulk* y asíncronas.
- Cuenta con 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.

A continuación se explicara la configuración del MC utilizada.

3.2.1 El oscilador y frecuencia de operación

El módulo USB del MC requiere de una frecuencia de reloj especifica para operar correctamente, específicamente 48 MHz son necesarios para operar en modo de velocidad máxima o 6 MHz cuando se va a utilizar en modo de baja velocidad. Para el desarrollo del módulo de comunicaciones USB se utilizo un cristal de 20 MHz como oscilador externo, derivándose el reloj interno del módulo PLL de 96MHz tal como se muestra en figura 26. Partiendo de esta frecuencia de reloj se configuró el MC para que funcionara en modo de máxima velocidad, generando una frecuencia de reloj interna de 48 MHz (equivalente a 12 mil instrucciones por segundo (MIPS)).

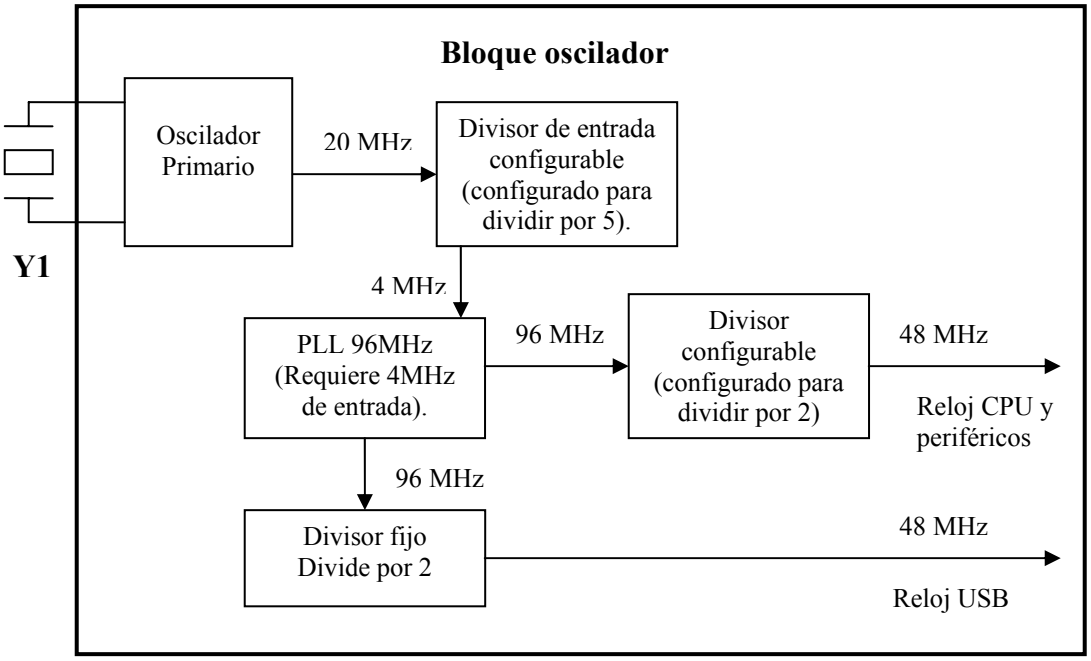


Figura 26 Configuración del reloj.

3.2.2 Alimentación

El MC opera a un voltaje de 5V. Este voltaje se puede obtener directamente del bus USB (un mínimo de 100mA están siempre disponibles en el bus, hasta un máximo de 500mA) o de una fuente de alimentación externa, como el MC cuenta con la tecnología nanoWat® es posible que se pueda alimentar con corrientes pequeñas por lo que se optó por alimentarlo del bus USB y no de una fuente externa. También se obtiene la ventaja al alimentarlo del voltaje del bus USB que se desacopla de la alimentación externa del circuito, con lo que se evita retroalimentación de corriente entre los circuitos externos y la PC quedando el

paciente aislado de corrientes altas y por consiguiente no existe el riesgo de choques eléctricos al paciente.

3.2.3 Interfaz USB

Para el circuito se utilizaron el regulador de voltaje, transmisor, receptor y resistencias *pull-up* internas del MC. Esto ayudo a reducir el número de componentes externos del circuito. La conexión USB puede ser desconectada eléctricamente deshabilitando el módulo USB en el *firmware*. Para deshabilitar el módulo en el *firmware* poner a '0' el bit USBEN en el registro UCON [22], con esto el regulador USB integrado también se deshabilitara, es decir, se simula la desconexión física del cable USB al bus. La figura 27 muestra la conexión eléctrica del circuito.

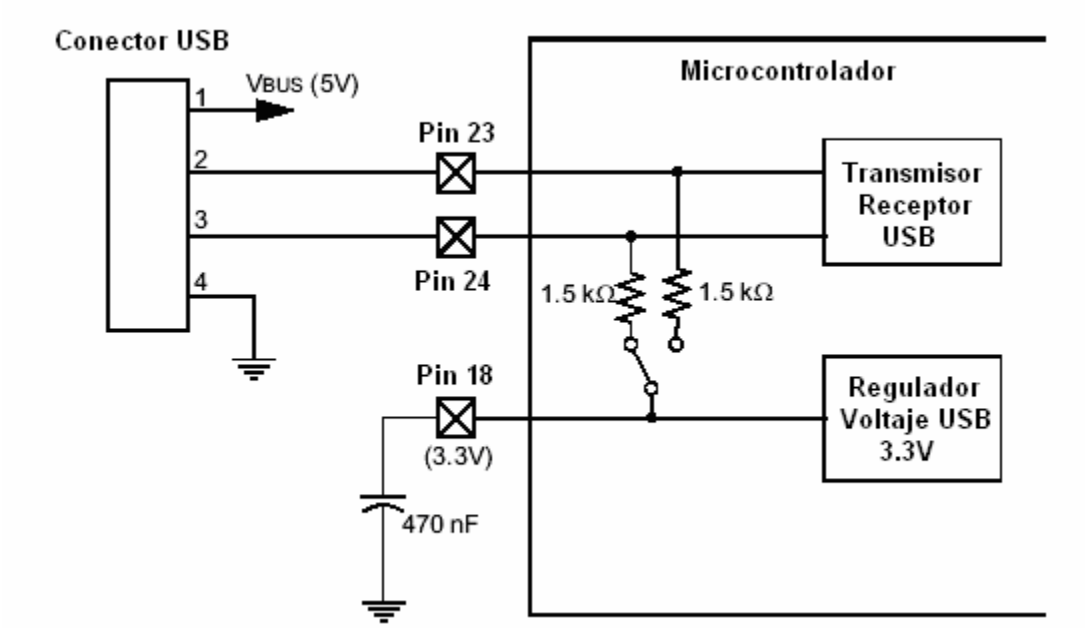


Figura 27 Interfaz USB, los pines corresponden al MC.

3.3 Estructura del *Firmware* (FW)

La estructura del FW USB es un sistema de archivos construido de tal forma que pueda ser usado para crear nuevas aplicaciones USB. Esto gracias al diseño estructurado del proyecto que contiene el código necesario para el funcionamiento del protocolo USB y un espacio para colocar el código propio de cada usuario. La totalidad del proyecto se encuentra localizado dentro de un solo directorio raíz, con varios subdirectorios que contienen el código fuente y aplicaciones.

Para el desarrollo de este FW se utilizaron las siguientes herramientas

- MPLAB IDE® versión 7.11, que es el compilador y *debugger* de Microchip®.
- Microchip C18® versión 2.42, que es el equivalente de lenguaje C para el MC.
- 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.

La estructura del proyecto consiste en una colección de subdirectorios y archivos específicos dentro de un directorio raíz del proyecto. El directorio raíz para el proyecto puede ser puesto en cualquier parte dentro el disco duro de una computadora y tener cualquier nombre de carpeta valido, sin embargo, la estructura de los subdirectorios se debe mantener siempre. Los siguientes subdirectorios y archivos necesitan estar siempre presentes.

Subdirectorios:

- **_output** Contiene los archivos de salida (FW compilado).
- **autofiles** Contiene la configuración global del USB y sus descriptores.
- **system** Contiene el FW del USB.
- **user** Recipiente para desarrollos posteriores.

Archivos:

- **CleanUp.bat** Limpia la carpeta de salida (_output) y todos los subdirectorios de compilaciones anteriores del código.
- **io_cfg.h** Mapea los nombres de los pines del MC a nombres de funciones, este archivo puede ser modificado para activar o desactivar pines del MC.
- **main.c** Este archivo contiene el `main ()` de la aplicación.
- **MCHPUSB.mcp** Archivo de proyecto utilizado por el MPLAB IDE®.
- **MCHPUSB.mcw** Archivo de área de trabajo utilizado por el MPLAB IDE®.

3.3.1 Estructura lógica

La estructura del FW USB provee un conjunto de interfases modulares que manejan la mayor parte del trabajo de la comunicación USB la figura 28 muestra un flujo típico del FW USB. Cada referencia al FW esta escrita para tener un ambiente cooperativo, de esta forma, no es necesario implementar funciones de bloqueo del flujo del programa.
En la función `main ()` existe un ciclo infinito "`while`" que maneja diferentes tareas. Estas pueden pensarse lógicamente como una tarea USB o una tarea de usuario. Las tareas USB son manejadas por `USBDriverService ()`, la cual sondea y atiende todas las interrupciones USB. Cuando una transacción de control del punto de destino o ENDPOINT es recibida, se llama a `USBCtrEPService ()`. Todas las transferencias sobre el

endpoint de control por defecto, necesitan seguir el protocolo de transferencia tal y como esta descrito en la especificación del protocolo USB

El servicio de transferencia de control esta proporcionado por las funciones contenidas en el archivo `usbctrltrf.c`. El primer caso de todas las transferencias de control entrantes es una petición. Una petición USB puede ser a cualquiera de dos estándares o a una clase especifica. Una petición estándar es atendida por `USBCheckStdRequest ()`, que maneja las peticiones solicitadas tal y como esta especificado en el capítulo 9 del estándar USB 2.0 [42]. Una petición a una clase específica necesita ser manejada por el archivo del FW que sabe que hacer con ese servicio. Un ejemplo de estas clases son los dispositivos de interface humanos (*Human Interface Device* o HID) y las clases de dispositivos de comunicación (*Communication Device Class* o CDC) el ejemplo de la figura 29 muestra HID, este podría referirse fácilmente a cualquier otra clase de dispositivo. Los manejadores de una petición a una clase específica necesitan estar contenidos en el archivo dentro del FW específico como por ejemplo `hid.c` o `cdc.c`. El nombre dado a una función que atiende una petición especifica es `USBCheck<nombre de la clase>Request()`, donde `<nombre de la clase>` puede ser cualquier nombre especifico de la clase.

El proceso de enumeración USB es manejado principalmente en `usb9.c`. Uno de los pasos más importantes en la enumeración de procesos es el manejo de la petición SET_CONFIGURATION, que es hecha por `USBStdSetCfgHandler ()`. Esta función puede ser modificada por el usuario para llamar a las funciones apropiadas para inicializar el *endpoint* de la aplicación. Cada dispositivo específico de la clase del FW necesita tener una rutina de inicialización del *endpoint*. La convención utilizada para los nombres de estas funciones son `<nombre de la clase>InitEP()`, donde `<nombre de la clase>` puede ser cualquier nombre de la clase del dispositivo. Se debe tener cuidado de verificar que el índice de configuración, esta siendo consultado para enviarse al *host* USB. Un dispositivo puede tener múltiples configuraciones, y no todas las interfaces pueden ser la misma debido a una configuración diferente. El código de las aplicaciones del usuario es llamado también por la función `main ()`, y dicho código se encuentra localizado por defecto dentro de la función llamada `ProcessIO ()`. Cuando una aplicación necesita enviar o recibir una transacción USB, puede llamar una función de pre-escritura que agrega funcionalidad a los servicios de enviar y recibir declarados en el FW, tales como `HIDRxReport ()` y `HIDTxReport ()`.

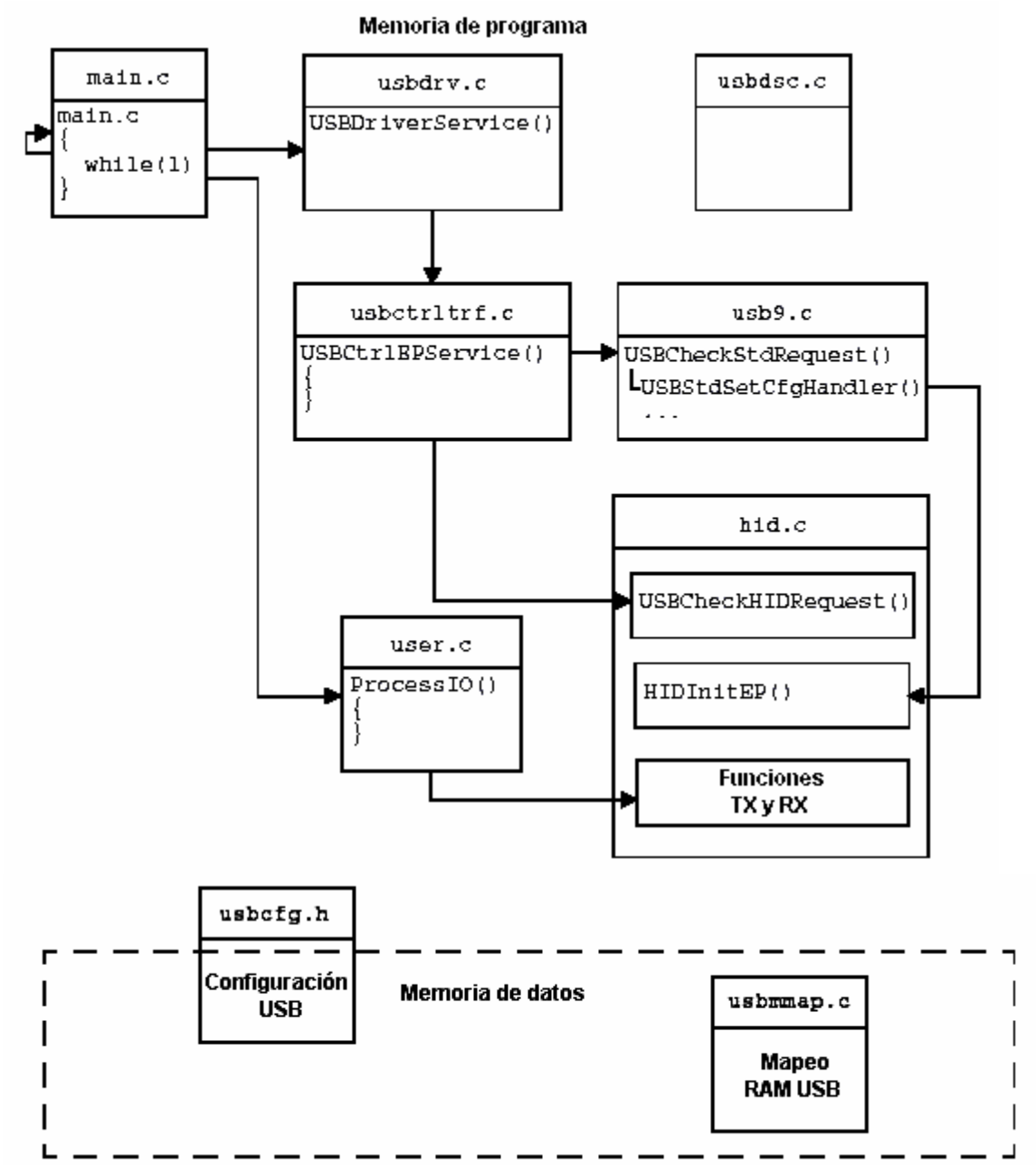


Figura 28 Relaciones entre archivos de la estructura lógica y la aplicación.

La configuración de dispositivos USB es manejada también en forma modular, modificando variables en un pequeño número de archivos; la información se encuentra de esta forma de manera global en tiempo de compilación. Los archivos son altamente interdependientes, y transfieren información entre ellos mismos durante el tiempo de compilación para crear la configuración completa del USB. Las interrelaciones se muestran en la figura 29.

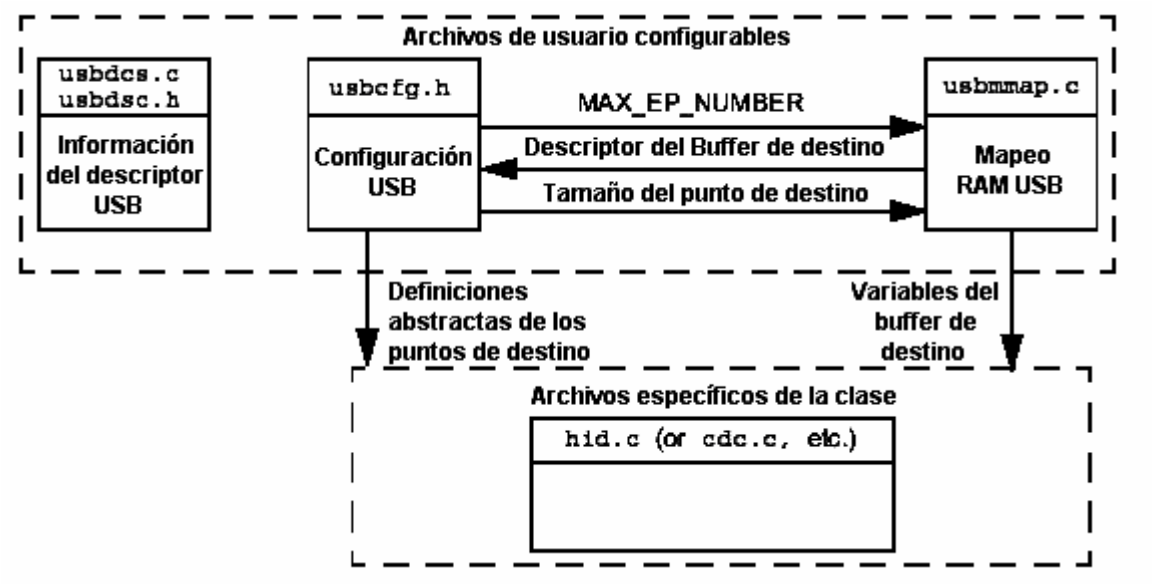


Figura 29 Relaciones en tiempo de compilación entre archivos de configuración USB.

3.4 El FW USB en la estructura lógica.

Los subdirectorios `autofiles` y `system` contienen el código fuente del FW USB dentro la estructura lógica. Algunos de esos archivos puede que nunca sea necesario modificarlos, mientras que otros necesitan ser editados para configurar la aplicación. Cada fólder y sus archivos serán discutidos a continuación a fin de saber que hace cada uno de ellos.

3.4.1 Directorio `autofiles`.

Este directorio contiene tres archivos cruciales para la operación del USB:

- `usbcfg.h` Archivo de configuración global del USB.
- `usbdsc.c` Archivo descriptor del USB.
- `usbdsc.h` Archivo de cabecera del descriptor USB.

3.4.1.1 `usbcfg.h`

Este archivo de cabecera es el archivo más importante del proyecto entero. Define el mapeo de los *endpoints* a sus funciones de clase. También define el número de identificación (Microchip USB ID (MUID)), el cual es usado para determinar a cual clase del USB pertenece actualmente el cambio de mando en progreso. Finalmente, define el tamaño máximo del buffer para el *endpoint* 0.
Para centralizar la configuración de características importantes del USB, todas las opciones de tiempo de compilación relacionadas con el USB se encuentran en este archivo. El

archivo esta compuesto como una serie de directivas de compilador `#define`. Las opciones para configurar son:

- `EP0_BUFF_SIZE[buffer_size]`
- `MAX_NUM_INT [max]`
- `MODE_PP [_PPBMn]`
- `UCFG_VAL [option1 | option2...]`
- `USE_SELF_POWER_SENSE_IO`
- `USE_USB_BUS_SENSE_IO`
- `USB_USE_GEN or USB_USE_CDC or USB_USE_HID`
- `MAX_EP_NUMBER [max_ep]`

`EP0_BUFF_SIZE` Define el tamaño del buffer para el *endpoint* 0. Puede tener un valor valido de 8, 16, 32 o 64 bytes esta definición es usada globalmente en el proyecto para muchas cosas. Es usada durante la compilación del proyecto para reservar el tamaño apropiado del buffer para el *endpoint* 0. Es usada por el descriptor USB para informar al receptor o *HOST* USB del tamaño del buffer del *endpoint* 0. Es usado también durante la operación de transferencia de control.
Cuando se defina esta variable, tomar en cuenta que un dispositivo USB de baja velocidad solamente puede usar un buffer de 8 bytes, mientras que uno de velocidad total puede usar buffer de 8, 16, 32 o 64 bytes.

`MAX_NUM_INT` Define el tamaño del arreglo de la configuración activa de cada interfase, que puede ser cambiada durante la operación, los valores validos son enteros [0, 1, 2…] si un dispositivo tiene múltiples configuraciones, el número de interfases para la configuración con el número mayor de interfases debe ser usado.
Por ejemplo, un dispositivo con dos configuraciones tiene tres interfases en la primera y dos interfases en la segunda. En este caso, `MAX_NUM_INT` necesita ser tres.

`MODE_PP` Define el buffer del modo ping-pong a ser usado durante la ejecución. Los valores permitidos son `_PPBM0`, `_PPBM1` y `_PPBM2`. Cada uno de ellos se encuentra definido en el archivo de cabecera `usbdrv.h`. (Ver registro `UCFG` sección 17.4.4 del *datasheet* del MC).

`UCFG_VAL` Define el valor inicial de el registro especial `UCFG`. Los valores permitidos son:

- `_LS` o `_FS` selecciona entre velocidad baja o velocidad total.
- `_TRINT` o `_TREXT` selecciona entre el modo de transmisión-recepción interno o externo.
- `_PUEN` se usan los resistores *pull-up* internos del MC.
- `_OEMON` se usa el indicador de salida SIE (*Serial Interface Engine*).
- `_UTEYE` habilita el modo de salida *eye-pattern* (Ver la sección 17.2.2.7 del *datasheet* del MC).

Las opciones pueden ser encadenadas, por ejemplo:

```
#define UCFG_VAL _PUEN | _TRINT | _FS
```

Estos valores son usados por la estructura lógica del FW para inicializar el registro `UCFG`.

Nota

1. Si la opción `_LS` es seleccionada, la configuración del reloj necesita ser cambiada para soportar velocidad baja.
2. La opción `_UTEYE` es usada solo para probar el MC, y nunca en aplicaciones. Cuando es habilitado, el MC transmite continuamente. NO conectar el MC a un huésped USB cuando esta opción esta habilitada.

`USE_SELF_POWER_SENSE_IO` Indica que el MC esta detectando la presencia de una alimentación *on-board* a través de un pin de entrada-salida. Si la tarjeta sobre la que se usa el MC no usa un pin de entrada-salida para detectar la presencia de voltaje, esta definición necesita ser comentada.

`USE_USB_BUS_SENSE_IO` Indica que el FW puede usar el pin definido en `io_cfg.h` para determinar cuando se ha activado el módulo USB, si el diseño de la tarjeta no usa un pin de entrada-salida para detectar la presencia del bus USB, esta se puede comentar.

Cuando `USE_USB_BUS_SENSE_IO` no esta definida, el módulo USB esta siempre habilitado. Usar esta característica permite mejorar la eficiencia de energía del sistema debido a que el módulo USB solo se activa cuando se conecta al bus USB.

`USB_USE_GEN, USB_USE_CDC y USB_USE_HID` son usados para indicar que clases USB necesitan ser incluidas en el código del proyecto. Cuando cada una de ellas esta definida, le dice al archivo de cabecera USB global `usb.h` que archivos de cabecera de clase específicos necesitan ser incluidos.
El archivo de cabecera `usb.h` es usado globalmente como un archivo necesario cuando se compila el proyecto, si la clase HID es usada, entonces, `hid.c` y `hid.h` necesitan ser agregadas al proyecto. Si la clase CDC es usada, entonces `cdc.c` y `cdc.h` se agregan.

Esta definición es usada también cuando se recibe una petición `SET_CONFIGURATION`. El usuario puede cambiar la "sección modificable" en la función `USBStdSetCfgHandler` localizada en `usb9.c`. El código localizado en esta sección inicializa los *endpoints* usados por cada clase USB específica, que se encuentre mapeada en `usbcfg.h`.

`MAX_EP_NUMBER` necesita ser igual al número más grande de *endpoints* usados en el proyecto. Por ejemplo, si el mayor número de *endpoints* es 5, entonces `MAX_EP_NUMBER` necesita ser 5. Esta definición es usada principalmente en `usbmap.c` para reservar el buffer de los registros descriptores de USB.

3.4.1.2 usbdsc.c y usbdsc.h

Estos archivos contienen la información del descriptor USB para el MC. El archivo principal, usbdsc.c, define al descriptor en si mismo, usbdsc.h define la estructura del descriptor, que es usada para calcular el tamaño del descriptor con la función sizeof (). Cuando un descriptor es agregado o removido de la configuración del descriptor principal, (por ejemplo CFG01), el usuario solo tiene que cambiar la estructura del descriptor definida en usbdsc.h.
Una configuración típica de un descriptor consiste de los siguientes componentes:

- Al menos una configuración del descriptor (USB_CFG_DSC).
- Uno o más descriptores de interfase (USB_INTF_DSC).
- Uno o más descriptores de *endpoints* (USB_EP_DSC).

3.4.1.2.1 Personalizando USBDSC.C

En la versión del archivo usbdsc.c, incluida con la estructura lógica del FW indica que parámetros necesitan ser definidos, y puede servir como una plantilla para desarrollar nuevos descriptores. La mayoría de los archivos se encuentran comentados en el manual de programación del FW, pero vale la pena hacer referencia a los más importantes.

USB_CFG_DSC El atributo de configuración necesita siempre tener la definición _DEFAULT como mínimo. Dos opciones adicionales, _SELF y _RWU, pueden ser encadenadas con _DEFAULT. _SELF indica que el *host* USB de el MC está auto alimentado, mientras que _RWU indica que el *host* USB soporta *wakeup* remoto, las definiciones de estas opciones se encuentran dentro de usbdefs_std_dsc.h.

USB_EP_DSC Un descriptor para un *endpoint* tiene una forma similar a:

```
Sizeof (USB_EP_DSC) , DSC_EP , _EP01_OUT , _BULK , 64 , 0x00
```

Los primeros dos parámetros especifican la longitud del descriptor (7) y el tipo de descriptor. El siguiente parámetro identifica el *endpoint*, que tiene el formato _EP<##>_<dir> donde ## es el número del *endpoint* y dir es la dirección de transferencia (IN o OUT). El siguiente parámetro identifica el tipo de *endpoint*. Las opciones disponibles son _BULK, _INT, _ISO y _CTRL (para *endpoints Bulk*, interrupción, asíncronos y de control respectivamente). _CTRL no se usa por que no se encuentra definido en los descriptores USB. Cuando _ISO es usado, se le pueden encadenar opciones adicionales. Por ejemplo _ISO | _AD | FE lo cual describe a un *endpoint* como una tubería sincrona y de reacción adaptiva. Ver usbdefs_std_dsc.h y la especificación del protocolo USB para mas detalle. Los parámetros finales definen el tamaño máximo del *endpoint* y el intervalo de muestreo.

La cadena del descriptor USB Parece más bien como una simple línea de texto, el descriptor está formado por una estructura de datos particular. El arreglo de datos del descriptor tiene el formato:

```
rom struct{ byte bLength;byte bDscType;Word string [size]; }
sdxxx={ sizeof (sdxxx) , DSC_STR , <text>} ;
```

Esta estructura provee un mecanismo para que el compilador de C18 pueda calcular la longitud de caracteres del descriptor `sdxxx`, donde `xxx` es el número. Los primeros dos bytes del descriptor son la longitud y el tipo de descriptor. El texto restante `<text>` son caracteres en formato Unicode. La cadena de texto restante es declarada como un arreglo de palabras con el número de caracteres igual a `<size>`; el cual debe ser calculado manualmente contando los caracteres. Por ejemplo, si la cadena es "USB", entonces el descriptor de la cadena necesita ser `(índice usado 002)`:

```
rom struct{ byte bLength;byte bDscType;word string[3]; }
sd002= { sizeof (sd002) , DSC_STR , 'U' , 'S' , 'B'} ;
```

Un proyecto USB puede tener múltiples caracteres. El FW soporta el manejo de varios caracteres a través de una tabla de búsqueda, que esta definida como:

```
rom const unsigned char *rom USB_SD_Ptr[] = {&sd000, &sd001,
&sd002};
```

El ejemplo anterior tiene 3 cadenas de caracteres (sd000, sd001, sd002). Las cuales pueden ser agregadas o removidas como sea necesario. Las cadenas son manejadas por la tabla de búsqueda `USB_SD_Ptr`; el índice de la cadena corresponde a la posición del índice dentro del arreglo (`&sd000` esta en la posición `USB_SD_Ptr [0]`, etcétera…). La tabla de búsqueda `USB_SD_Ptr` es usada por la función `USBStdGetHandler ()` dentro de `usb9.c`. `sd000` es un descriptor especializado, define el código del idioma, que es generalmente US *English* (0x0409).

3.4.1.2.2 Personalizando `USBDSC.H`

En el archivo de encabezado `usbdsc.h`, las variables para cada componente del descriptor son nombradas con las siguientes convenciones:

- **`USB_CFG_DSC`** los tipos son llamados `cdxx`, donde `xx` es el número de configuración. Este número debe coincidir con el valor del índice actual de la configuración.
- **`USB_INTF_DCS`** los tipos son llamados `i<yy>a<zz>`, donde `yy` es el número de la interfaz y `zz` es el número de interfaz alternativo.
- **`USB_EP_DSC`** los tipos son llamados `ep<##><d>_i<yy>a<zz>`, donde `##` es el número del *endpoint* y `d` es la dirección de transferencia. El nombre de interfaz

también debería ser puesto en una lista como un sufijo para identificar a que interfaz del *endpoint* pertenece.

El siguiente ejemplo muestra la estructura en `usbdsc.h` de la configuración de un descriptor. Este MC tiene una configuración con dos interfaces, interfaz 0 con 2 *endpoints* (entrada y salida) y una interfaz 1 con un *endpoint* (entrada). La jerarquía de los descriptores mostrada sigue los requerimientos del protocolo USB. Todos los *endpoints* pertenecientes a una interfaz necesitan ser listados inmediatamente después de la interfaz.

```
#define CFG01 rom struct
{    USB_CFG_DSC    cd01;
     USB_INTF_DSC   100a00;
     USB_EP_DSC     ep01o_100a00;
     USB_EP_DSC     ep01i_100a00;
     USB_INTF_DSC   101a00;
     USB_EP_DSC     ep02i_101a00;
}cfg01
```

3.4.1.2.3 Agregando configuraciones

Un dispositivo USB puede tener más de una configuración de descriptor, por ejemplo `CFG02`, `CFG03`, etc.… Para agregar otra configuración de un descriptor, es necesario implementar un nuevo conjunto de estructuras, similares a `CFG01`, en `usbdsc.c` y `usbdsc.h`. Estas pueden ser llamadas `CFG02` o `CFG03` etc.… Cuando esto se ha hecho, agregar el nombre de la nueva configuración del descriptor a la tabla de búsqueda `USB_CD_Ptr`, en el mismo método usado para manejar las cadenas de los descriptores. `USB_CD_Ptr [0]` es un número ficticio para la configuración 0, el estado no definido dentro de la especificación del protocolo USB. El manejador de las configuraciones `USBStdSetCfgHandler` también necesita ser modificado para soportar las configuraciones adicionales.

3.4.2 Directorio `system`

El directorio del proyecto `system` contiene muchas de las funciones del núcleo USB. Su estructura se muestra en la figura 30.

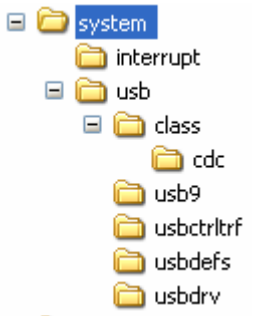


Figura 30 Directorio system.

Además de el manejador de memoria USB `usbmap.c`, los archivos discutidos en esta sección están localizados en las siguientes carpetas dentro de la carpeta `usb`.

- usb9: `usb9.c y usb9.h`
- usbctrltrf: `usbctrltrf.c y usbctrltrf.h`
- usbdrv: `usbdrv.c y usbdrv.h`
- usbdefs: los archivos de definición estándar `usbdefs_ep0_buff.h` y `usbdefs_std_dsc.h`.

3.4.2.1 *USBMMAP.C*

Este archivo es el manejador de memoria USB, permite en tiempo de compilador reservar memoria para los *endpoints* USB. Usa las opciones de tiempo de compilación especificadas en `usbcfg.h` para instanciar los *endpoints* y sus *buffers*.
Los *endpoints* son definidos usando el número de *endpoint* y la dirección de transferencia. Por simplicidad, `usbmmap.c` solo usa el número del *endpoint* en el registro de asignación `BDT`. Esto significa que si `usbcfg.h` declara que `MAX_EP_NUMBER` es 1, entonces 4 `BDT`s pueden ser instanciados: uno para *endpoint*0 de entrada y *endpoint*0 de salida que son siempre instanciados por el control de transferencia por *default*, y otro para *endpoint*1 de entrada y salida respectivamente.

El manejador de memoria USB usa `MAX_EP_NUMBER`, tal y como esta definida en `usbcfg.h`, para definir los *endpoints* a ser instanciados. Esto representa el número máximo de *endpoints*, no todos los *endpoints* son usados. Como los `BDT`s para los endpoints tienen direcciones asignadas por hardware en el banco 4, poner este valor muy alto resulta en un uso ineficiente de la memoria de datos RAM. Por ejemplo si una aplicación usa solamente los *endpoints* `EP0` y `EP4`, entonces `MAX_EP_NUMBER` es 4 y no 2. Entonces los `BDT`s de los *endpoints* intermedios (`EP1, EP2 y EP3`) no son usados, y 24 bytes de memoria asociada son desperdiciados (esto asumiendo el uso de un *buffer* ping-pong modo 0)

El nombre del *endpoint* instanciado es usado entonces en `usbcfg.h` para mapear el *endpoin* a esta función. Por ejemplo, supongamos una clase USB llamada "foo", que usa un *endpoint* de entrada y otro de salida, cada uno de 64 bytes, la definición de los *endpoints* para la clase foo seria:

```
#define FOO_INTF_ID     0x00
#define FOO_UEP         UEP1
#define FOO_BD_OUT      ep1Bo
#define FOO_BD_IN       ep1Bi
#define FOO_EP_SIZE     64
```

Los nombres son arbitrarios con el formato `FOO_????????`. Por abstracción, cualquier código escrito para el MC necesita usar las definiciones abstractas como `FOO_BD_OUT`. `FOO_BD_OUT` y no `ep1Bo` o `ep1Bi` Notar que el tamaño del *endpoint* definido en `usbcfg.h` es usado de nueva cuenta en `usbmap.c`. Esto demuestra que la relación entre ambos archivos es muy estrecha.

El buffer del *endpoint* para función USB necesita ser mapeado en el área de la memoria RAM de doble puerto (`0x400` a `0x7FF`) y tiene que estar después de que se insta ciaron las `BDTs`

3.4.2.2 *USBDRV.C*

Este archivo proporciona los servicios de bajo nivel para el protocolo USB, y maneja todas las interrupciones de *hardware*. A continuación se describen las funciones relevantes.

3.4.2.2.1 Función *USBCheckBusStatus*

Esta función deberá ser llamada una sola vez en la función `main ()`. Cuando un pin de entrada salida es usado para detectar le conexión o desconexión del cable USB, esta rutina habilita o deshabilita el módulo USB. Esto permite incrementar la eficiencia del sistema. Si el pin de entrada salida no se usa, el enunciado `#define USE_USB_BUS_SENSE_IO` en `usbcfg.h` necesita ser comentada. Esto causa que `USBCheckBusStatus ()` habilite el módulo USB después de que el MC es reseteado.

3.4.2.2.2 Función *USBDriverService*

Esta función sondea todas las banderas buscando una interrupción de hardware en los registros `UIR` y `UIE` para saber si un evento USB ha ocurrido, y los servicios del evento. También limpia la bandera de transacción USB completada, `TRNIF`, varios eventos pueden ocurrir cuando una aplicación hace transacciones de envió o recepción mas de cuatro veces antes de limpiar la bandera `TRNIF`. Esto ocurre por la cola FIFO de 4 niveles que almacena

transacciones completadas y puede ser vaciada solo un nivel a la vez limpiando `TRNIF`. Cuando más de cuatro transacciones son recibidas sin que se limpie `TRNIF`, no se pueden atender más transacciones. Un ejemplo de que es lo que pasa se muestra en el Ejemplo B, un poco más adelante. Este detiene el módulo USB para responder a cualquier petición de un huésped sin esperar que exista espacio en la pila.

La aplicación del usuario puede no limpiar directamente el bit `TRINF`, puede llamar a `USBDriverService ()`, como se muestra en los ejemplos A y C. limpiando el bit directamente si una transacción 0 ocurre.

Ejemplo A: Método típico de realizar una transacción por ciclo (pseudo código)

```
main ()
{
       while (1)
       {
              USBDriverService ();

              if (!mHIDTxTsBussy ())
                     HIDTxReport ();
       }//fin while
}//fin main
```

Ejemplo B: Método incorrecto de realizar múltiples transacciones por ciclo (pseudo código)

```
main ()
{
       while (1)
       {
              USBDriverService();

              while (mHIDTXIsBussy ());
              HIDTxReport ();             // 1 Tx
              while (mHIDTXIsBussy ());
              HIDTxReport ();             // 2 Tx
              while (mHIDTXIsBussy ());
              HIDTxReport ();             // 3 Tx
              while (mHIDTXIsBussy ());
              HIDTxReport ();             // 4 Tx
              while (mHIDTXIsBussy ());   // este nunca puede ser falso pues el
                                          // Buffer esta lleno.
              HIDTxReport ();             // 5 Tx código no ejecutado
       }//fin while
}//fin main
```

Ejemplo C: Método correcto de implementar múltiples transacciones por ciclo (pseudo código)

```
main ()
{
       while (1)
       {
              USBDriverService();

              while (mHIDTXIsBussy ());
              HIDTxReport ();            // 1 Tx
              USBDriverService ();
              while (mHIDTXIsBussy ());
              HIDTxReport ();            // 2 Tx
              USBDriverService ();
              while (mHIDTXIsBussy ());
              HIDTxReport ();            // 3 Tx
              USBDriverService ();
              while (mHIDTXIsBussy ());
              HIDTxReport ();            // 4 Tx
              USBDriverService ();
              while (mHIDTXIsBussy ());
              HIDTxReport ();            // 5 Tx
              USBDriverService ();

              //código de otra aplicación
       }//fin while
}//fin main
```

3.4.2.2.3 Función `USBSuspend`

Esta rutina primero suspende el módulo USB y habilita la interrupción de actividad del bus USB. Esta es una sección modificable donde se puede introducir código para crear un esquema de ahorro de energía. Antes de poner a dormir el MC, es necesario habilitar un código de activación del MC. Para poder configurar el bus USB a fin de que responda a una señal de habilitación del bus, es necesario activar el bit indicado en el ejemplo en el registro UIE.

Ejemplo Habilitando la suspensión del USB

```
PIR2bits.USBIF = 0;     //asegurarse que la bandera esta limpia
PIE2bits.USBIE = 0;     //activar wakeup del USB
Sleep();                //llama a sleep
PIR2bits.USBIF = 0;     //limpiar bandera
PIE2bits.USBIE = 0;     //deshabilitar interrupción USB
```

3.4.2.2.4 Función `USBWakeFromSuspend`

Esta rutina es llamada cuando una interrupción de actividad de bus es habilitada, deshabilita la interrupción de actividad del bus y habilita el módulo USB.

3.4.2.2.5 Función `USBRemoteWakeup()`

Si la aplicación soporta la función de *wakeup* remoto, al llamar a esta rutina se envía una señal de reanudar al *host*. Debe de ser llamada cuando un estimulo externo cause que se active el módulo USB. En el ejemplo se muestra el código que puede estar dentro de `USBSuspend ()`, la cual llama a `USBRemoteWakup ()`.

Ejemplo habilitando activación remota:

```
PIR2bits.USBIF = 0;          //asegurarse que la bandera esta limpia
INTCOMbits.RBIF = 0;         //asegurarse que la bandera esta limpia
PIE2bits.ISBIE = 1;          //fijar el código USB de activación
INTCOMbits.RBIE = 1;         //fijar el puerto para el código de activación

Sleep();                     //ir a seleep

If (INTCONbits.RBIF = = 1)   //revisar si el código de activación tiene puesta
                             // la interrupción.
{
       USBRemoteWakeup();    // si se cumple activo la función
}                            //fin if

PIR2bits.USBIF = 0;          //limpiar bandera
PIE2bits.USBIE = 0;          //deshabilito interrupción USB
INTCOMbits.RBIF = 0;         //limpiar bandera
INTCONbits.RBIE = 0;         //deshabilito interrupción del puerto
```

3.4.2.3 `USBCTRLTRF.C`

Este archivo maneja la transferencia de control, y ofrece servicio a todas las clases USB que necesitan manejar una petición USB. Como el control de cambio de servicios es compartido por muchas clases USB y la mayoría de las transacciones tienen múltiples operaciones, es importante saber que clase tiene actualmente el control de la sección de transferencia. Esto se logra utilizando un identificador para cada transferencia USB (`MUID`). Cuando `MUID` es igual a `MIUD_NULL`. Significa que ninguna clase esta manejando una petición. Si el manejador en `usb9.c` sabe atender una petición, la sesión de transferencia de control actual se actualiza a `MUID_USB9`. Igualmente si una petición de HID es recibida, entonces `MUID` puede ser `MUID_HID`.
Las declaraciones para `MUID` están definidas en `usbcfg.h`, y pueden variar de un proyecto a otro. Esto significa que dos proyectos diferentes pueden tener diferente declaración de `MUID`s.
Cada control de transferencia tiene tres estados Configuración, Datos y Estado (*Setup*, *Data* y *Status*). La maquina de estados para manejar los diferentes casos de una transferencia de control dentro el FW se ilustra en la figura.

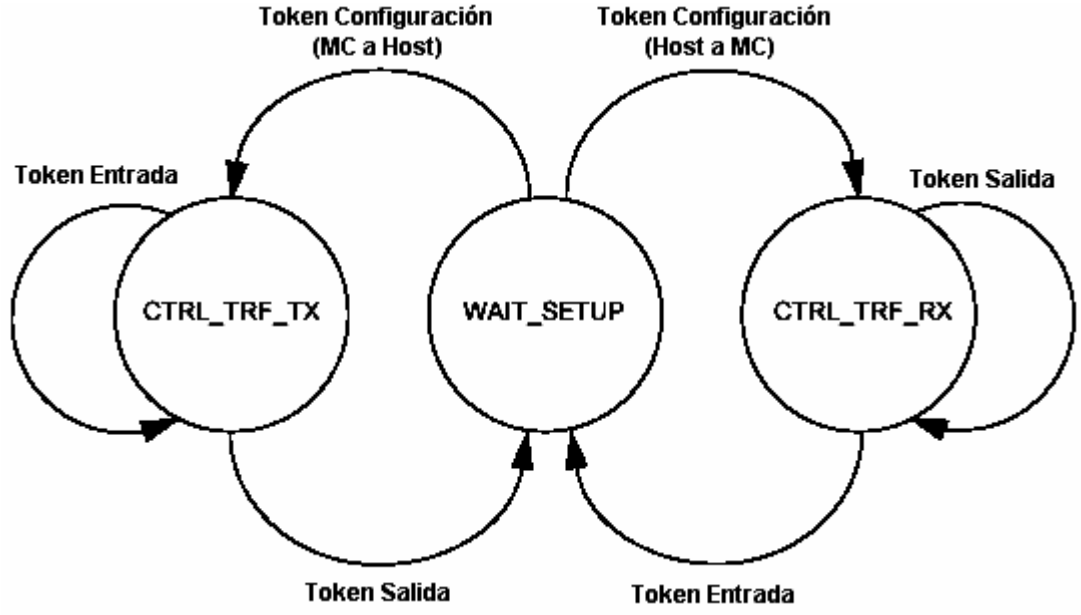


Figura 31 Maquina de cambio de estados

Observamos que cuenta con tres diferentes estados `WAIT_SETUP`, `CTRL_TRF_TX` y `CTRL_TRF_RX`.
`USBCtrlEPService ()` es llamada de `usbdrv.c`, solo servicios de transacciones pueden pasar por *endpoint*0, el cual checa por los diferentes tipos de transacciones para llamar al manejador apropiado (*Endpoint* 0 *SETUP*, *Endpoint* 0 *OUT* o *Endpoint* 0 *IN*).
Si el estado de la transferencia es *SETUP*, entonces `USBCtrlTrfSetupHandler ()` es llamada para atender el paquete. Cada paquete de control de transferencia USB siempre tiene 8 bytes de longitud. Estos son los pasos que se siguen para atender un paquete *SETUP*:

1. La rutina inicializa la maquina de control de estados.
2. llama a las clases buscando alguna que sepa manejar la petición de servicio.
3. Cuando todas las peticiones han analizado el paquete SETUP, la función `USBCtrlEPServiceComplete ()` checa la dirección de transferencia del estado para determinar que estado debe tener *endpoint* 0.

`USBCtrlEPServiceComplete ()` cubre las tareas restantes del paquete, su principal tarea es asignar el control correcto a los *endpoints* para cada situación dada.
Existen tres posibles resultados:

1. *Endpoint* 0 es detenido si la petición no se puede atender.
2. *Endpoint* 0 es configurado para transferir datos al *host* durante el estado de datos.
3. *Endpoint* 0 es configurado para recibir datos del *host* durante el estado de datos.

El estado de datos puede expandirse sobre múltiples transacciones USB. Es importante mantener una pista sobre el origen de los datos, el destino y el tipo de los mismos. Los cuales son rastreados y actualizados utilizando 4 variables dedicadas:

- pSrc
- pDst
- wCount
- el bit de memoria tipo bandera usb_stat.ctrl_trf_mem, que puede tener el valor de _ROM o _RAM.

Cuando se escriba una petición de manejo al control de transferencia de datos del MC al *host*, hay que asegurarse de lo siguiente:

1. Fijar el origen de los datos:
   - Para datos de la RAM: pSrc.bRam = <localización del buffer de RAM>;
   - Para datos de la ROM: pSrc.bRom = <localización de la memoria de programa>;

2. Fijar el tipo de origen:
   - usb_stat.ctrl_trf_mem = <_ROM o _RAM>;

3. Fijar el tamaño del dato a transferir:
   - wCount._word = < tamaño del dato a transferir>;

4. la dirección de destino del dato es manejada automáticamente por usbctrltrf.c; el destino es el *buffer* CtrlTrfData definido en usbmap.c.

Cuando se escribe una petición al manejador para recibir una transferencia de control de un *host* sobre múltiples transacciones, hay que asegurarse de hacer lo siguiente:

1. Fijar el destino de los datos:
   - pDst.bRam = <localizacion del buffer de RAM>

2. Inicializar el contador de recibidos:
   - wCount._word = 0

3. La dirección del origen de los datos en manejada automáticamente por usbctrltrf.c el destino es el *buffer* CtrlTrfData definido en usbmap.c.

3.4.2.4 USB9.C

Este archivo maneja las peticiones estándar USB tal y como están definidas en el capítulo 9 de la especificación USB, y maneja la numeración de los procesos. La función de interés de este archivo es USBStdSetCfgHandler (), que se debe modificar para hacer coincidir la configuración y la aplicación.

3.4.3 Validación en tiempo de compilación

Un conjunto de reglas puede fijarse para corregir las variables USB. Para habilitar la validación en tiempo de compilación, agregar la siguiente línea de código en uno de los archivos del proyecto:

```
#include "system\usb\usb_compile_time_validation.h"
```

Este archivo comprueba `EP0_BUFF_SIZE`, que tenga un valor admitido (8, 16, 32 o 64 bytes) reglas de validación adicionales pueden ser agregadas.

3.5 Emulación de Puerto RS-232 sobre USB

Una aplicación de Windows® ve una conexión física RS-232 como un puerto COM y se comunica con el dispositivo conectado usando las funciones *CreateFile*, *ReadFile* y *WriteFile*. Es debido a la simplicidad de esta forma de comunicación que se decidió implementar la emulación de un puerto RS-232 sobre la conexión USB, además de que no es necesario la utilización de *drivers* adicionales al sistema puesto que la comunicación con este tipo de puertos ya esta contemplada dentro del sistema operativo Windows®, y se logra utilizando los *drivers* `usbser.sys` y `ccport.sys` los cuales están soportados desde la versión de Windows®98 *Second Edition*.
La descripción del funcionamiento de dichas librerías queda fuera del alcance de este trabajo pero es posible obtener la información necesaria dentro del sitio Web de Microsoft®.
Una de las principales ventajas como ya se había mencionado es que no es necesario dar mantenimiento a *drivers* que soporten nuestra aplicación puesto que se utiliza un estándar del sistema operativo. Como el protocolo USB ya maneja los detalles de la comunicación de bajo nivel, el concepto de transferencia de *bauds* (Pulsos por segundo), bit de paridad y control de flujo para el estándar RS-232 se vuelve abstracto.

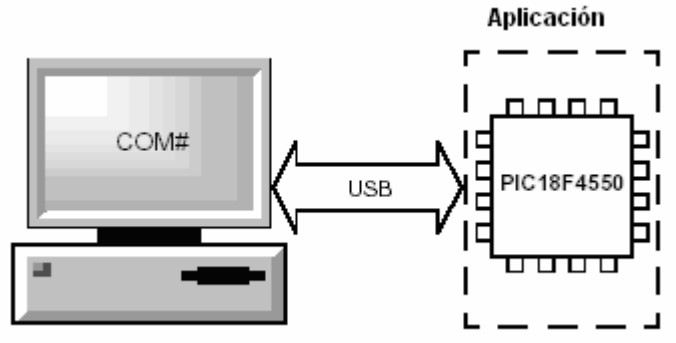


Figura 32 Emulación RS-232 sobre USB

3.5.1 Especificación CDC USB

La especificación de las clases del dispositivo de comunicación (CDC (*Communication Device Class*)) define varios modelos de comunicación, incluyendo la emulación serial. Todas las referencias a la especificación CDC en este documento se refieren a la versión 1.1 del *driver* de Microsoft® `usbser.sys` .

La especificación del CDC describe un modelo de control abstracto para la emulación serial sobre USB. En resumen, se requiere de dos interfases USB. La primera es la interfaz de la clase de comunicaciones, utiliza un *endpoint* de interrupción de entrada. Esta interfaz es utilizada para notificar al huésped USB del estatus de la conexión RS-232 para el dispositivo USB RS-232 emulado, la segunda es la interfaz de la clase de datos, utiliza un *endpoint* de salida tipo *Bulk* y un *endpoint* de entrada también tipo *Bulk*. Esta interfaz es usada para transferir bytes de datos limpios (sin encabezados o empaquetamientos) que normalmente son transferidos sobre la verdadera interfaz RS-232. Es decir no hay que preocuparse por programar descriptores o escribir manejadores de función para una clase especifica. Los descriptores para un dispositivo USB RS-232 emulado y los manejadores de la clase ya se encuentran definidos dentro el *firmware* del MC.

3.5.2 Peticiones y notificaciones especificas de la clase

Dentro del *firmware* se implementaron los siguientes manejadores de peticiones de la clase:

- `SEND_ENCAPSULATED_COMMAND`
- `GET_ENCAPSULATED_COMMAND`
- `SET_LINE_CODING`
- `GET_LINE_CODING`
- `SET_CONTROL_LINE_STATE`

3.6 Funciones USB UART

La selección de la función a utilizar depende de varios factores:

¿Termina la cadena de datos en un carácter nulo?.
¿Se conoce la longitud de la cadena de datos?.
¿El origen de los datos se encuentra en la memoria de programa o en RAM?.

Función	Descripción
putrsUSBUSART	Escribe un carácter terminado en nulo de la memoria de programa al USB.
putUSBUSART	Escribe un carácter terminado en nulo de la memoria de datos al USB.
mUSBUSARTTxRom	Escribe una cadena de caracteres de longitud conocida de la memoria de programa al USB.
mUSBUSARTTxRam	Escribe una cadena de caracteres de longitud conocida de la memoria de datos al USB.
mUSBUSARTTxTrfReady	¿Esta listo el dispositivo para recibir una cadena nueva del USB?
getsUSBUSART	Leer una cadena de caracteres del USB.
mCDCGetRxLength	Leer la longitud de la última cadena leída del USB.

4 Software de usuario

En capítulos anteriores se mostró el diseño y la construcción del *hardware* que registra las señales fisiológicas, en particular las señales PPG, ahora bien, dado que dicho *hardware* esta construido para que envíe la lectura mediante un puerto USB a una computadora personal o de escritorio, es necesario construir un *software* que sea capaz de obtener, graficar y analizar los datos generados a partir del *hardware* mencionado.

En este capitulo se muestra el diseño, la arquitectura y la constitución del sistema, así como los diagramas de clases y los diagramas de secuencias relacionados con la construcción del sistema.

4.1 Diseño del sistema StressHunter3

En esta sección describiremos el diseño del sistema StressHunter3, comenzando por la descripción general del sistema, se continúa con la descripción de los paquetes o submódulos, posteriormente se revisan los diagramas de clases, seguido de los diagramas de secuencias para finalizar con los detalles de la aplicación.

4.1.1. Descripción del sistema

El sistema tiene como objetivo principal el manejo y administración de las señales fisiológicas, en especifico, la señal PPG registradas por paciente, además de realizar un análisis matemático, de acuerdo a los datos establecidos, para que un medico calificado pueda realizar un diagnostico en base a los resultados del análisis. Para ello se requiere de sistema que sea amigable y fácil de usar para cualquier persona.

Así pues, el sistema realiza fundamentalmente los siguientes procesos:

- ***Dar de alta pacientes***: el sistema es capaz de agregar nuevos pacientes para registrarlos en la base de datos y de esta manera llevar un registro de las señales fisiológicas que son generadas por dicho pacientes.

- ***Buscar un paciente existente***: el sistema de igual manera debe ser capaz de buscar los pacientes dentro de la base de datos para seleccionar uno de ellos y con ello, asignarle una nueva medición de señales fisiológicas PPG o bien revisar las evaluaciones anteriores de dicho paciente.

- ***Medición y graficación de señales fisiológicas para un paciente anónimo***: para aquellas mediciones que no requieran ser guardadas en la base de datos, el sistema tiene la capacidad de generar la graficación de las señales PPG y de generar un análisis sin tener que asignarle dichos valores a un paciente en especial.

- ***Medición y graficación de señales fisiológicas para un paciente existente***: una vez que se ha localizado un paciente el sistema es capaz de obtener las evaluaciones previas del paciente en cuestión, o bien de realizar una nueva medición para analizarla y finalmente almacenarla en el historial del paciente.

4.2 Composición general del software de usuario.

La construcción del software de usuario fue dividido en varios paquetes que agrupan las diferentes funcionalidades de las clases programadas. Así pues, el sistema StressHunter3 esta compuesto de 8 paquetes los cuales se pueden ver en la figura 33 y se explican a continuación:

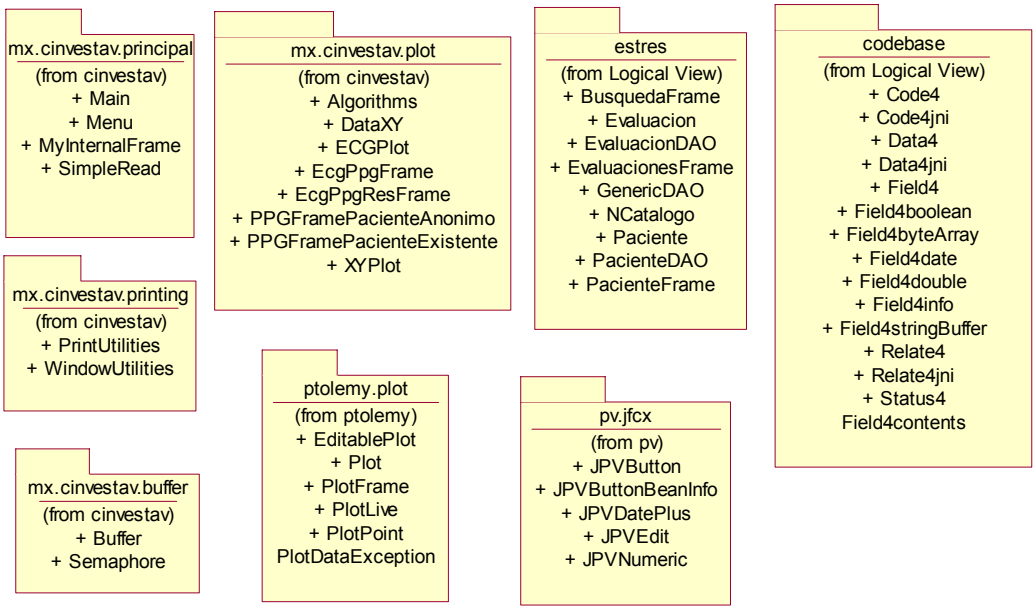


Figura 33 Diagrama que muestra el conjunto de paquetes que componen al sistema StressHunter3.

- **mx.cinvestav.principal:** dentro del paquete se encuentran la clase principal del sistema StressHunter3, además de otras 3 clases las cuales son usadas por la clase principal para crear el formulario inicial de la aplicación.

- **mx.cinvestav.plot:** dentro de este paquete se encuentran las clases que se encargaran de manejar la graficación de las señales PPG recibidas.

- **estres:** en este paquete se encuentran las clases que serán la interfase entre el usuario y la base de datos.

- **codebase:** con este paquete se realiza el manejo de la BD de una manera más transparente.

- **mx.cinvestav.printing:** dentro de este paquete se encuentran las clases con las que se hace uso de la impresora para imprimir las gráficas y los datos recibidos.

- **mx.cinvestav.buffer:** es el paquete en donde se encuentran las clases necesarias para la creación y manejo de un buffer de datos.

- **ptolemy.plot:** paquete que contiene las clases de la librería utilizada para realizar las graficas.

- **pv.jfcx:** paquete que contiene los controles necesarios para los formularios.

4.3 Diagramas de clases del software de usuario

En esta sección se mostrará de manera concreta la constitución del software StressHunter3, para ello solo serán mostradas y explicadas las clases que desempeñan las funciones más importantes dentro del sistema.

Primeramente se puede observar en la figura 34 la composición principal del sistema, en donde podemos ver que esta compuesto básicamente por cinco clases, en donde una es la clase principal y a partir de ella se construye la aplicación; las otras cuatro clases son las que construyen y manejan los formularios de la aplicación.

A continuación se dará una breve descripción de cada una de las clases mostradas en el diagrama de la figura 34 y además se detallará de manera general la composición de cada una de estas clases.

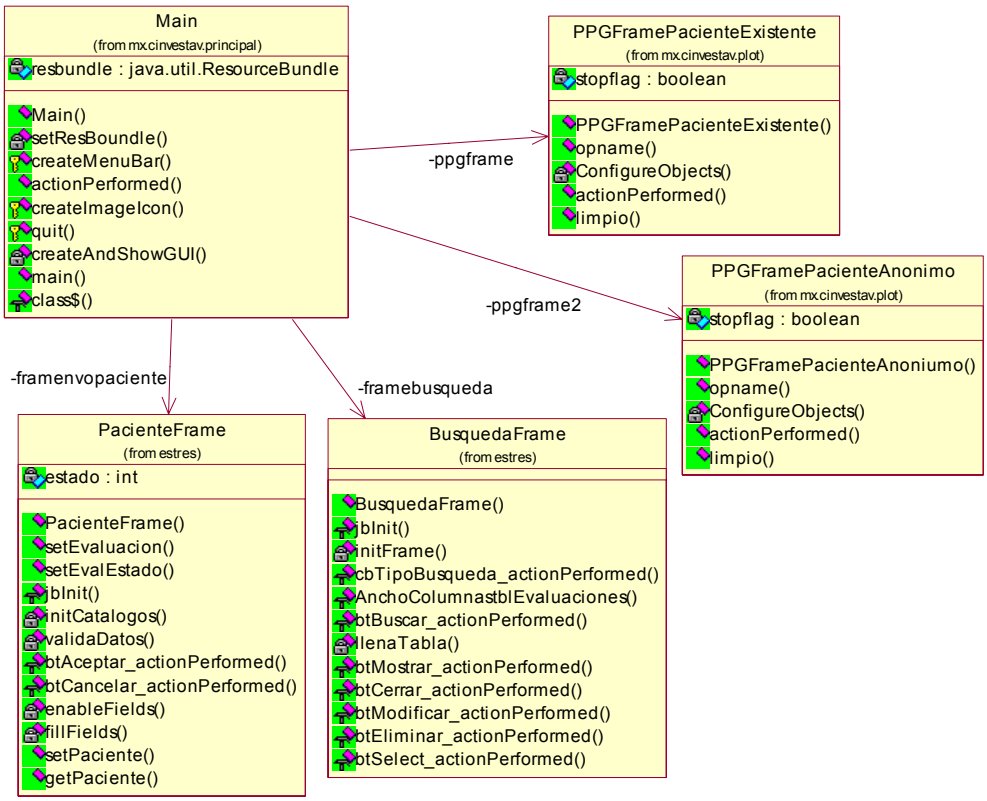


Figura 34 Diagrama de clases principal

- **Main:** Es la clase principal y es con la que da inicio el sistema, básicamente es la ventana principal que se ve al iniciar. En ella se cargan las demás ventanas y se inicializa el controlador del puerto COM para su posterior manejo.

- **PacienteFrame:** Es la clase encargada de mostrar el formulario con el cual se dan de alta los nuevos pacientes dentro del sistema. La composición de esta clase puede verse en la figura 35 y una breve descripción de las clases que la conforman se da a continuación:

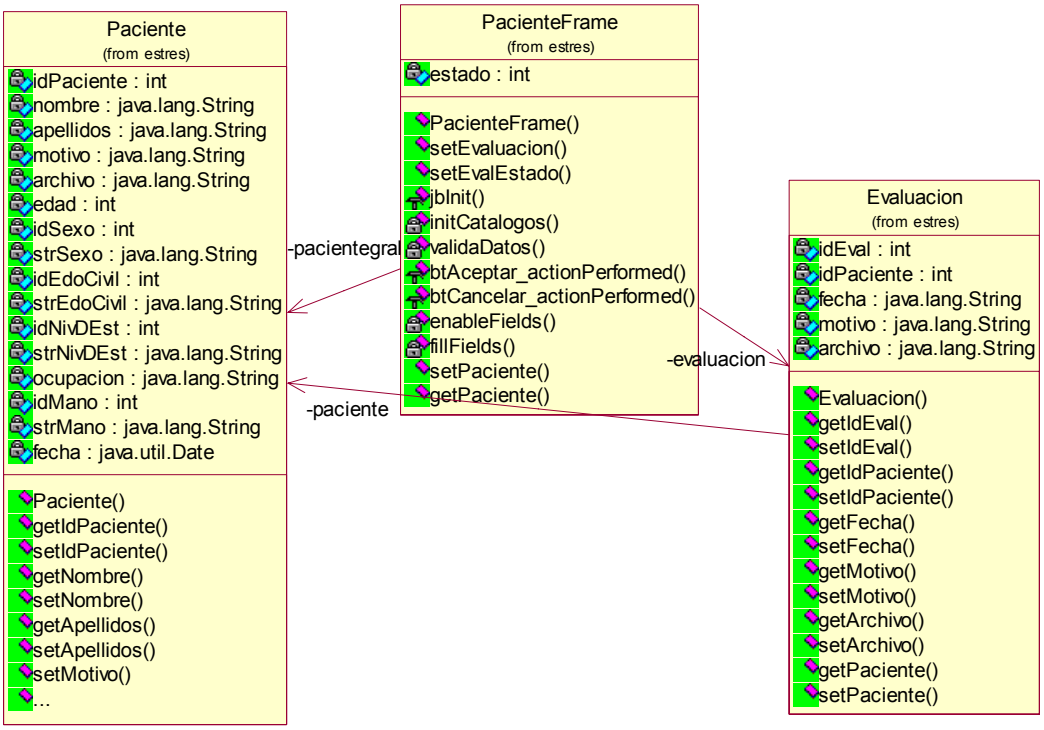


Figura 35 Diagrama de clases de la composición de la clase PacienteFrame.

- **Paciente:** Es la clase que es utilizada como objeto transporte entre el usuario y la base de datos para establecer los valores del nuevo paciente en la BD.

- **Evaluacion:** Es la clase que es utilizada para como objeto transporte entre el usuario y la base de datos para establecer los valores de una nueva evaluación de un paciente en la BD.

- **BusquedaFrame:** Esta clase tiene como objetivo el mostrar un formulario con el cual puedan realizarse las búsquedas de los pacientes existentes en la BD, para con ello obtener su información y agregar nuevas mediciones.

  - **Paciente:** Es la clase que es utilizada para como objeto transporte entre el usuario y la base de datos para establecer los valores recuperados de la búsqueda de un paciente en la BD.

  - **Evaluacion:** Es la clase que es utilizada para como objeto transporte entre el usuario y la base de datos para establecer los valores recuperados de la búsqueda de las evaluaciones de un paciente en la BD.

En la figura 36 se muestra el diagrama de clases que muestra la composición de la clase BusquedaFrame.

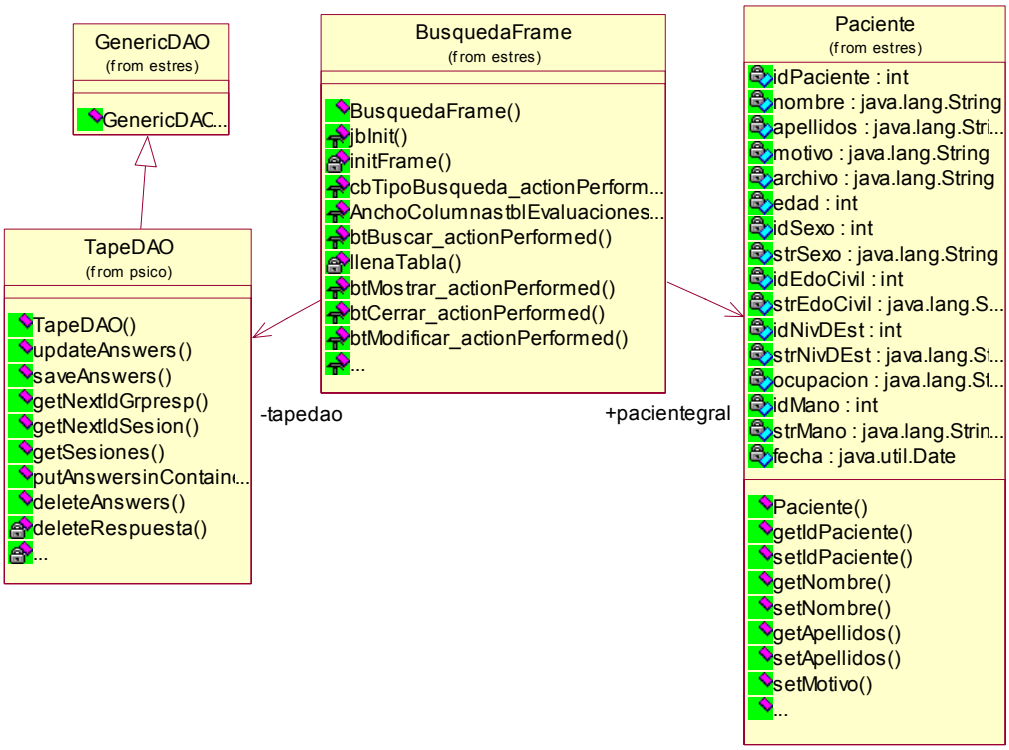


Figura 36 Diagrama de clases de la composición de la clase BusquedaFrame

- **PPGFramePacienteAnonimo:** Con esta clase se realiza el proceso de captura y graficación de la señal PPG de un nuevo paciente, sin la necesidad de que el paciente exista en la BD. Las clases que la componen se explican a continuación:

    - **SimpleRead:** Esta es la clase es la encargada de leer los datos provenientes de la tarjeta de captura y escribirlos en la clase buffer para su posterior graficación.

    - **Buffer:** Clase que maneja los datos leídos con la clase SimpleRead y establece los métodos necesarios para que la clase encargada de graficar reciba los datos y los procese sin ninguna perdida de los mismos.

    - **XYPlot:** Clase encargada de graficar los datos almacenados en la clase Buffer. Con esta clase es posible manipular tanto los datos como la visualización de los mismos.

    - **DataXY:** Esta clase almacena los datos graficados, además de los puntos establecidos por el usuario para con ello realizar el análisis de las señales fisiológicas registradas.

En la figura 37 se muestra el diagrama de flujo que muestra la composición de la clase **PPGFramePacienteAnonimo**, con las clases previamente explicadas.

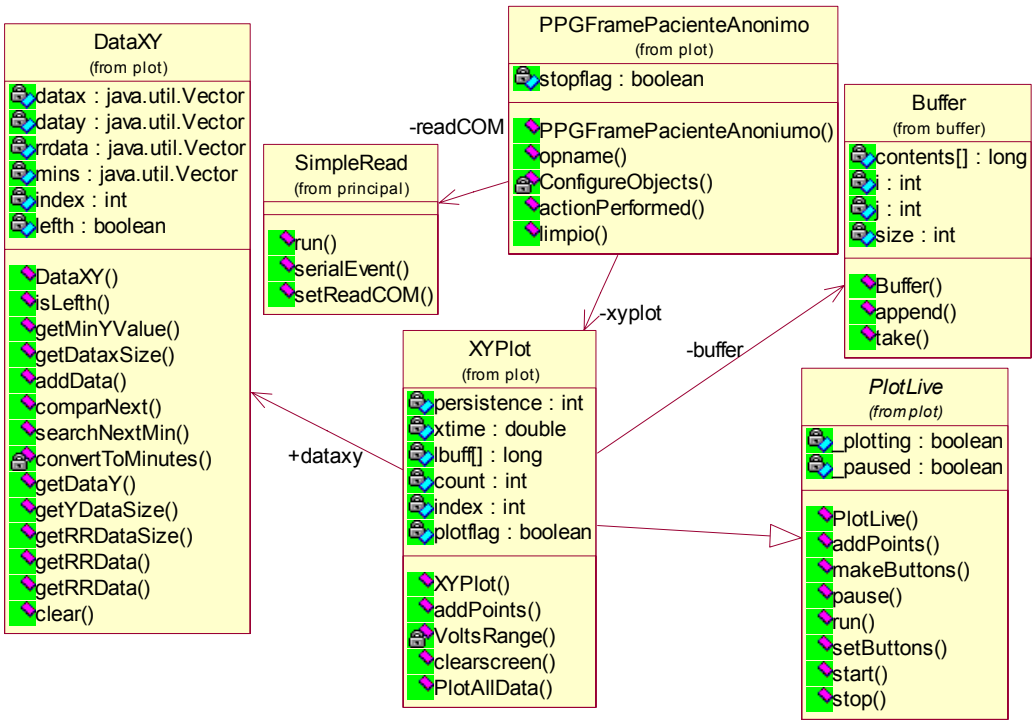


Figura 37 Diagrama de clases de la composición de la clase PPGPacienteAnonimo

- **PPGFramePacienteExistente:** Con esta clase el sistema logra capturar y graficar una señal de PPG, además de asignarla a un paciente, previamente localizado con la clase de **BusquedaFrame**. Las clases que componen a esta clase se explican a continuación:

    - **SimpleRead:** Esta es la clase es la encargada de leer los datos provenientes de la tarjeta de captura y escribirlos en la clase buffer para su posterior graficación.

    - **Buffer:** Clase que maneja los datos leídos con la clase SimpleRead y establece los métodos necesarios para que la clase encargada de graficar reciba los datos y los procese sin ninguna perdida de los mismos.

    - **XYPlot:** Clase encargada de graficar los datos almacenados en la clase Buffer. Con esta clase es posible manipular tanto los datos como la visualización de los mismos.

- **DataXY:** Esta clase almacena los datos graficados, además de los puntos establecidos por el usuario para con ello realizar el análisis de las señales fisiológicas registradas.

- **Paciente:** Es la clase que es utilizada para como objeto transporte entre el usuario y la base de datos para establecer los valores recuperados de la búsqueda de un paciente en la BD.

- **Evaluacion:** Es la clase que es utilizada para como objeto transporte entre el usuario y la base de datos para establecer los valores recuperados de la búsqueda de las evaluaciones de un paciente en la BD.

En la figura 38 se muestra el diagrama de flujo que muestra la composición de la clase **PPGFramePacienteAnonimo**, con las clases previamente explicadas.

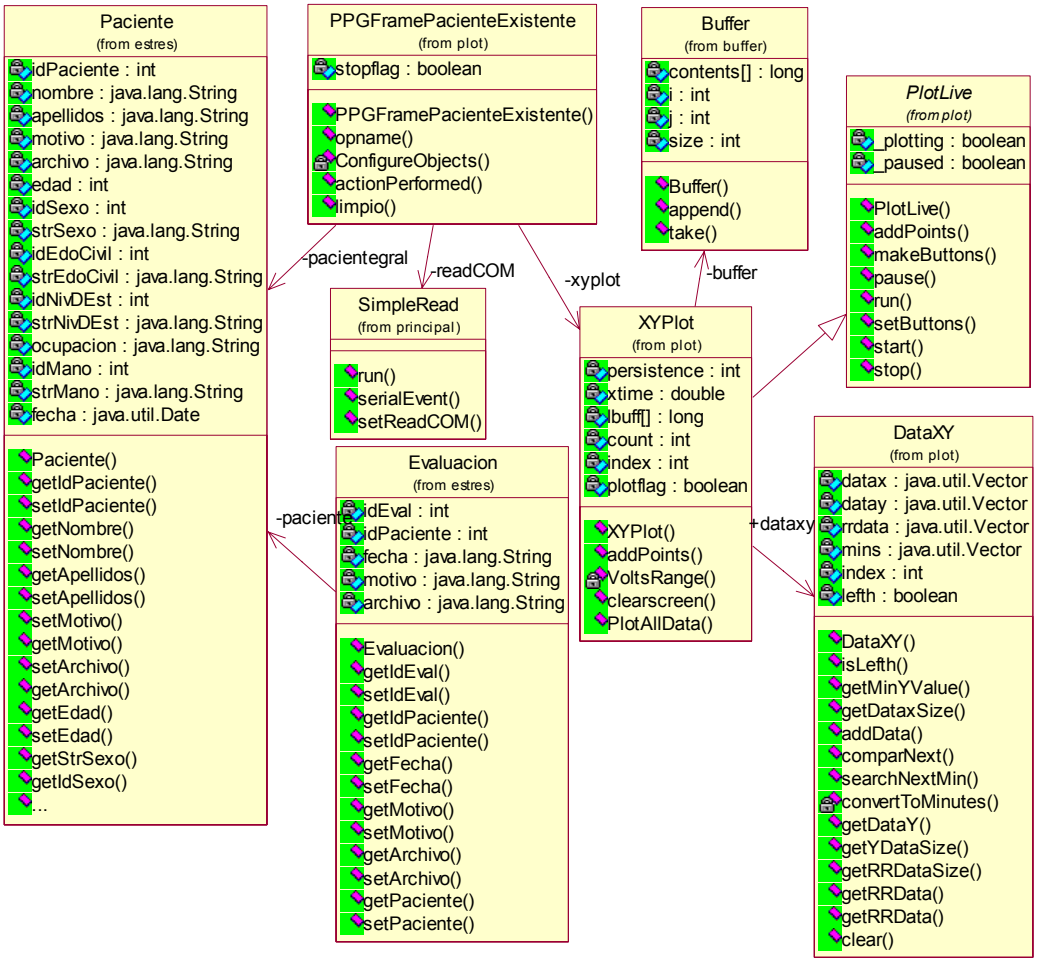


Figura 38 Diagrama de clases de la composición de la clase PPGFramePacienteExistente

4.4. Diagramas de secuencias

4.4.1 Crear un nuevo paciente

Cuando el usuario desea crear un nuevo paciente dentro del sistema StressHunter3, la secuencia que deben seguir las clases explicadas en la sección 4.2 es la que se puede ver en la figura 39 y a continuación se describen:

1. El usuario pulsa en el menú principal la opción de “paciente nuevo”.

2. Se crea una instancia de la clase PacienteFrame es creada y en ese momento se solicita la creación de un objeto de la clase PacienteGeneral

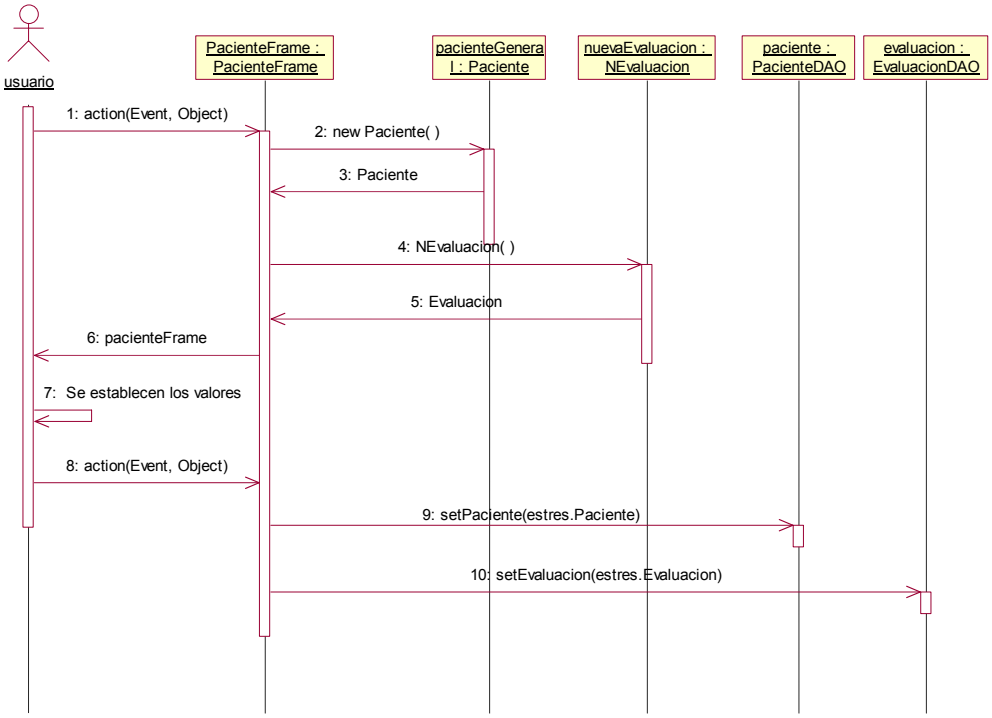


Figura 39 Diagrama de secuencia del proceso de paciente nuevo

3. Se crea una instancia de la clase PacienteGeneral

4. Se solicita una instancia de la clase nuevaEvaluacion

5. Se crea una instancia del objeto de la clase nuevaEvaluacion

6. Se crea una instancia del objeto de la clase PacienteFrame y se despliega el formulario al usuario.

7. El usuario establece los datos en el formulario para el nuevo paciente

8. Realiza una acción en donde le indica al formulario que guarde los datos registrados.

9. PacienteFrame hace un llamado a la clase PacienteDAO y guarda los valores del objeto pacienteGeneral en la base de datos.

10. PacienteFrame hace un llamado a la clase EvaluacionDAO y guarda los valores del objeto pacienteGeneral en la base de datos.

4.4.2 Búsqueda de un paciente

Cuando el usuario desea buscar un paciente dentro del sistema StressHunter3, la secuencia que deben seguir las clases explicadas en la sección 4.2 es la que se puede ver en la figura 40 y a continuación se da una breve descripción:

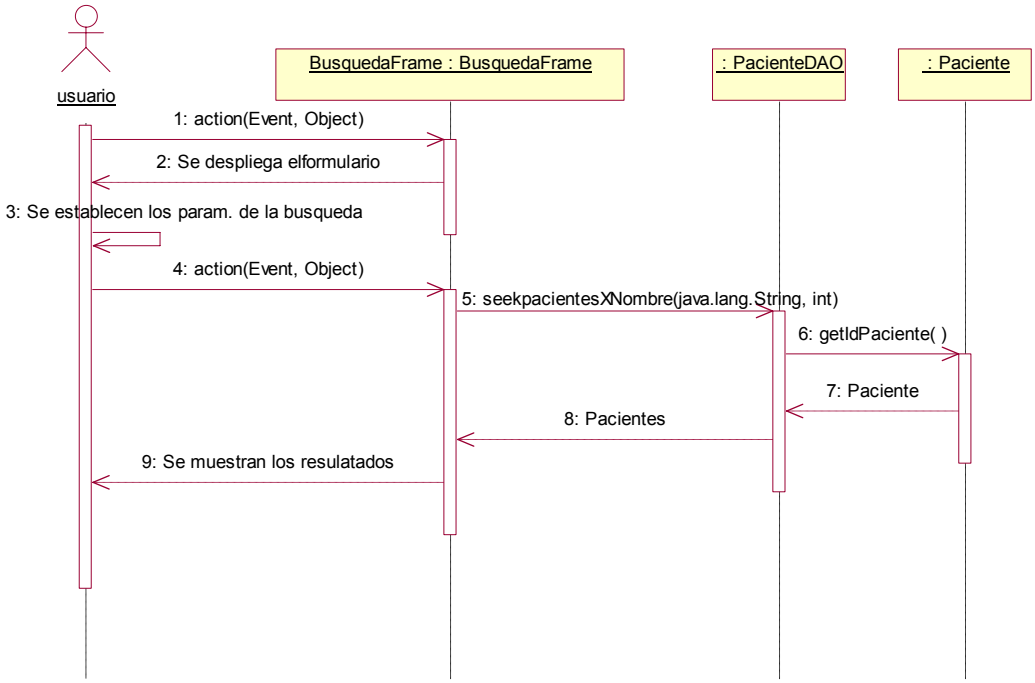


Figura 40 Diagrama de secuencia del proceso de busqueda de pacientes

1. El usuario pulsa en el menú principal la opción de “buscar paciente”.

2. Se crea y se despliega usuario el formulario de búsqueda creado con la clase BusquedaFrame.

3. El usuario establece los parámetros de la búsqueda.

4. El usuario pulsa el botón de búsqueda y genera la acción.

5. La clase BusquedaFrame le solicita a la clase PacienteDAO que realice la búsqueda en la base de datos por el nombre del paciente.

6. De acuerdo a la búsqueda PacienteDAO se solicita la creación de objetos PacienteGeneral.

7. Se generan tantos objetos PacienteGeneral como resultados encuentra de acuerdo a la búsqueda establecida.

8. Una vez que termina el proceso de búsqueda, PacienteDAO devuelve un objeto List de objetos pacienteGeneral al objeto BusquedaFrame

9. Se despliegan los resultados de los pacientes encontrados al usuario.

4.4.3 Graficación de la señal PPG de un paciente anónimo

Cuando el usuario desea graficar la señal PPG de un paciente anónimo dentro del sistema StressHunter3, la secuencia que deben seguir las clases explicadas en la sección 4.2 es la que se puede ver en la figura 41; y a continuación se da una breve descripción:

1. El usuario pulsa en el menú principal la opción de “PPG anónimo”.

2. Se establecen los parámetros de lectura para la señal PPG y comienza la lectura.

3. El usuario indica la acción de la graficación de la señal PPG.

4. PPGFrame le solicita a la clase XYPlot que comience la lectura de los datos.

5. Se ejecuta el método de simpleRead, el cual se encarga de leer los datos del hardware.

6. Se escriben los datos leídos en una instancia de la clase buffer.

7. Se limpia la pantalla de graficación.

8. Se toma un dato del buffer para graficarlo.

9. Se obtiene un objeto point.

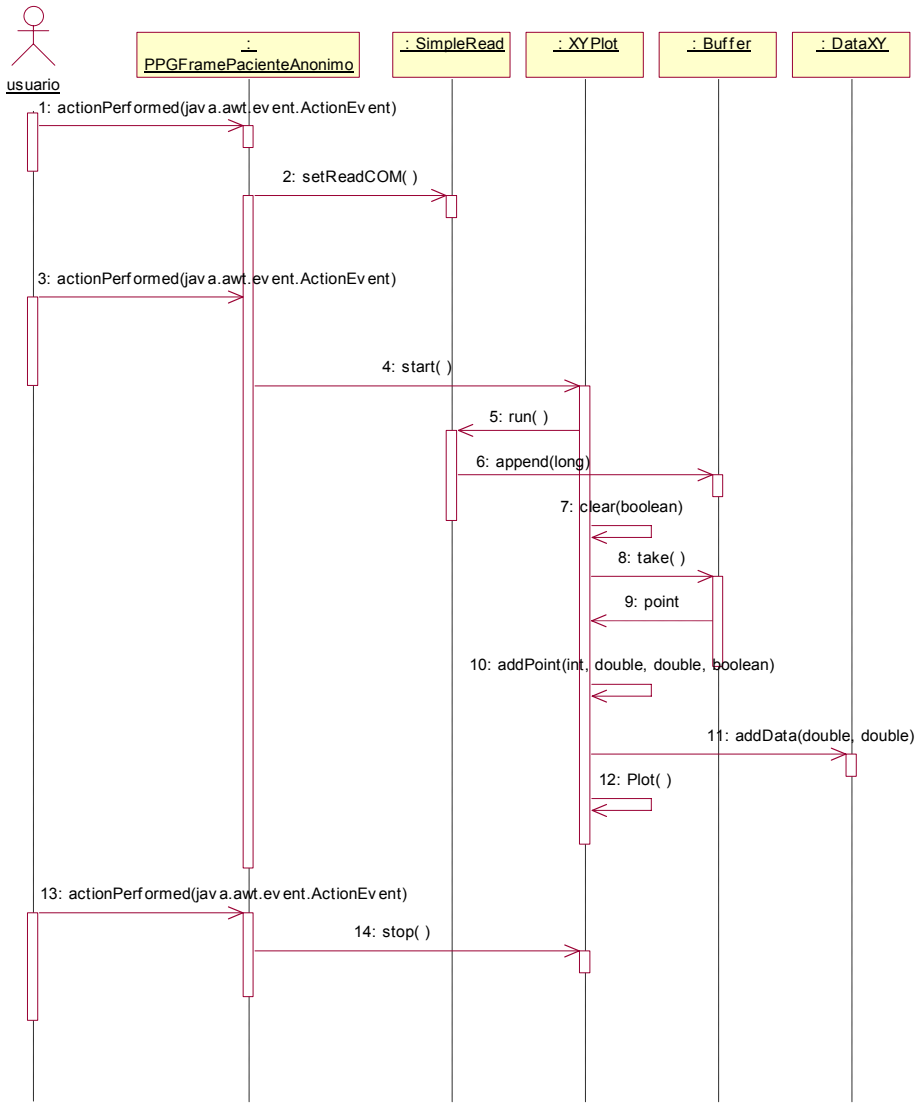


Figura 41 Diagrama de secuencia del proceso de graficación de señal PPG de un paciente anónimo

10. Se agrega el valor del objeto point a los datos que serán graficados.
11. Se agrega el valor del objeto point a los datos que serán analizados.
12. Se grafica el valor agregado.
13. El usuario decide en que momento detener el proceso y genera la acción.
14. Se detiene el proceso de graficación

4.4.4 Graficación de la señal PPG de un paciente existente

Cuando el usuario desea graficar la señal PPG de un paciente existente dentro del sistema StressHunter3, la secuencia que deben seguir las clases explicadas en la sección 4.2 es la que se puede ver en la figura 42 y a continuación se da una breve descripción:

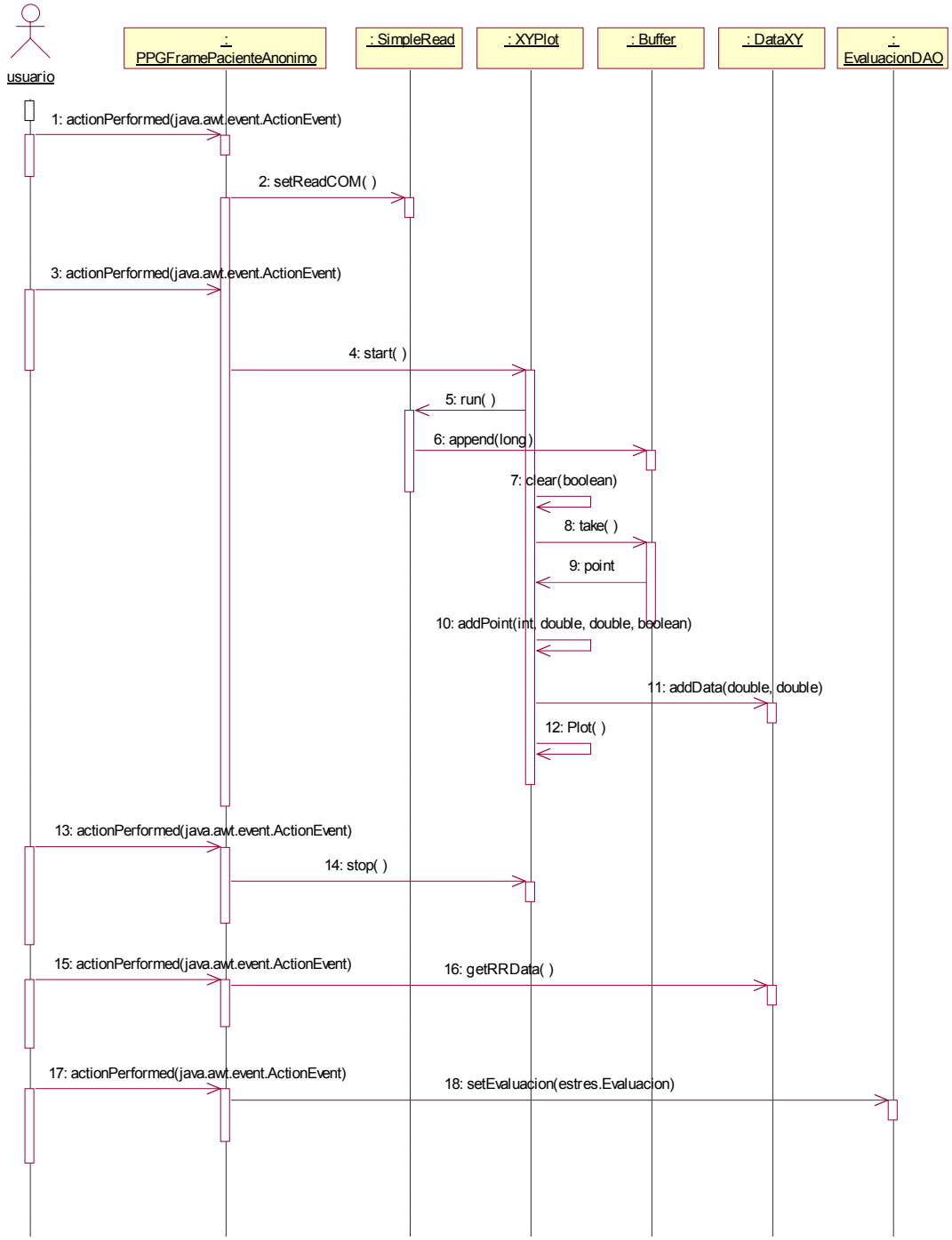


Figura 42 Diagrama de secuencia del proceso de graficación de señal PPG de un paciente existente

1. El usuario pulsa en el menú principal la opción de “PPG paciente registrado”.
2. Se establecen los parámetros de lectura para la señal PPG y comienza la lectura.
3. El usuario indica la acción de la graficación de la señal PPG.
4. PPGFrame le solicita a la clase XYPlot que comience la lectura de los datos.
5. Se ejecuta el método de simpleRead, el cual se encarga de leer los datos del hardware.
6. Se escriben los datos leídos en una instancia de la clase buffer.
7. Se limpia la pantalla de graficación.
8. Se toma un dato del buffer para graficarlo.
9. Se obtiene un objeto point.
10. Se agrega el valor del objeto point a los datos que serán graficados.
11. Se agrega el valor del objeto point a los datos que serán analizados.
12. Se grafica el valor agregado.
13. El usuario decide en que momento detener el proceso y genera la acción.
14. Se detiene el proceso de graficación
15. El usuario selecciona los puntos de control de la señal PPG realiza la acción para realizar el análisis.
16. El objeto DataXY efectúa el análisis de la señal y despliega los resultados.
17. El usuario realiza la acción para salvar la evaluación
18. La clase EvaluacionDAO es la encargada de guardar la evaluación generada de acuerdo al paciente establecido.

4.5 Detalles de la implementación del sistema StressHunter3

La implementación realizada del sistema StressHunter3 tiene las siguientes peculiaridades:

- El sistema fue desarrollado utilizando JSDK1.5.
- Como formato de base de datos fue utilizado el formato DBF.
- Para el manejo de base de datos en Java se utilizó la API de CodeBase para java.
- Para el manejo de las graficas de las señales PPG se utilizó la librería Plot.
- El sistema trabaja con un puerto COM simulado sobre el estándar USB 2.0 para la recolección de los datos de las señales fisiológicas PPG.
- Las evaluaciones de las señales fisiológicas PPG son almacenadas en formato XML para un manejo fácil.

5 Integración del proyecto

En este capítulo se explicará la manera en que se integraron cada una de las partes del proyecto explicadas en los capítulos anteriores, los archivos o librerías que se utilizan como enlace entre cada parte del proyecto, así mismo se mostrará el circuito impreso y su distribución funcional.

5.1 Diagrama de bloques

En capítulos anteriores se explicó el funcionamiento de los siguientes bloques:

- **Sensor de PPG,** revisado en el capítulo 2**,** explica el funcionamiento e integración del *hardware* de adquisición de datos, solo lo relacionado a la conversión, tratamiento y amplificación de la señal capturada por el sensor.
- **Transmisor USB**, revisado en el capítulo 3, explica solamente la implementación del módulo de transmisión USB y la conversión analógico-digital de la señal.
- **Software de Usuario**, revisado en el capítulo 4, explica el funcionamiento de la interfaz de usuario que muestra gráficamente los datos capturados, hace algunos análisis a la señal y administra los pacientes dentro de una base de datos así como las capturas de datos pertenecientes a cada uno de ellos.

De esto podemos obtener el diagrama de bloques de la figura 43.

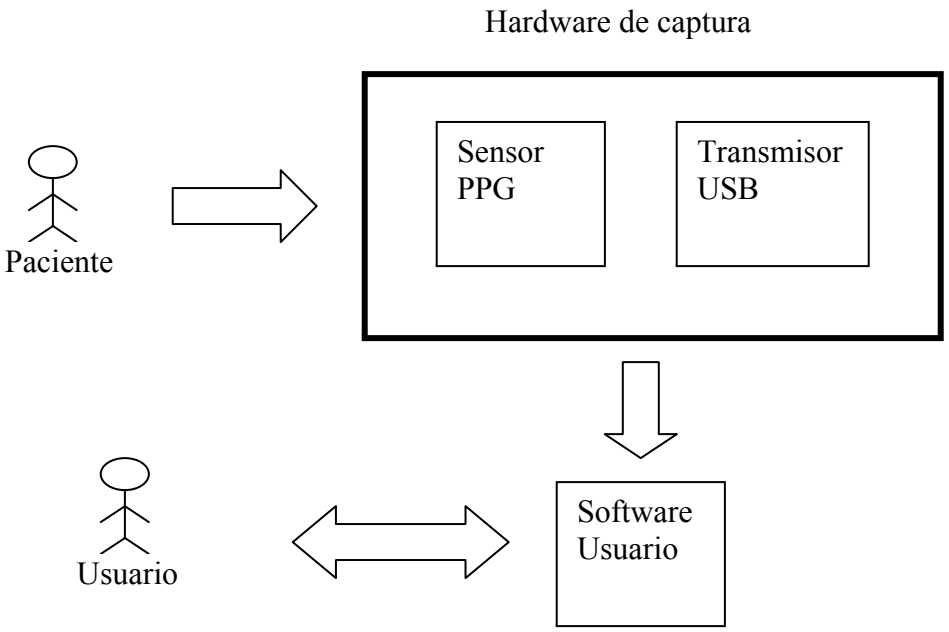


Figura 43 Diagrama de bloques.

Donde:

- **Paciente** es la persona a la que se conecta el dispositivo de captura de señales fisiológicas, que se encarga de enviar a través de un cable la señal analógica al *hardware* de captura.

- **Hardware de captura** contiene en una placa de circuito impreso el *hardware* correspondiente al sensor de PPG y el transmisor USB, se encarga de recibir la señal analógica del dispositivo de captura de señales, procesarla, convertirla en su equivalente digital y enviarla a la PC utilizando un cable USB.

- **Software de usuario** es el programa almacenado en la PC que interactúa con el usuario, ya sea un doctor o alguna persona encargada de realizar un estudio al paciente conectado al sistema. Dicho *software* se encarga de procesar los datos enviados por el *hardware* de captura y mostrarlos al usuario de una manera gráfica o numérica, permitiendo almacenar las mediciones para un estudio o comparación posterior.

- **Usuario** es un doctor o alguna persona encargada de realizar un estudio al paciente conectado al sistema e interactúa con el *software* de usuario interpretando los datos mostrados por este.

5.2 Circuito impreso

Podemos observar que los bloques del sensor de PPG y el transmisor USB forman un solo bloque, esto es por que se integraron en una sola tarjeta a fin de hacer el circuito lo más pequeño posible a continuación se muestra en la figura 44 y 45 la primer versión funcional del sistema.

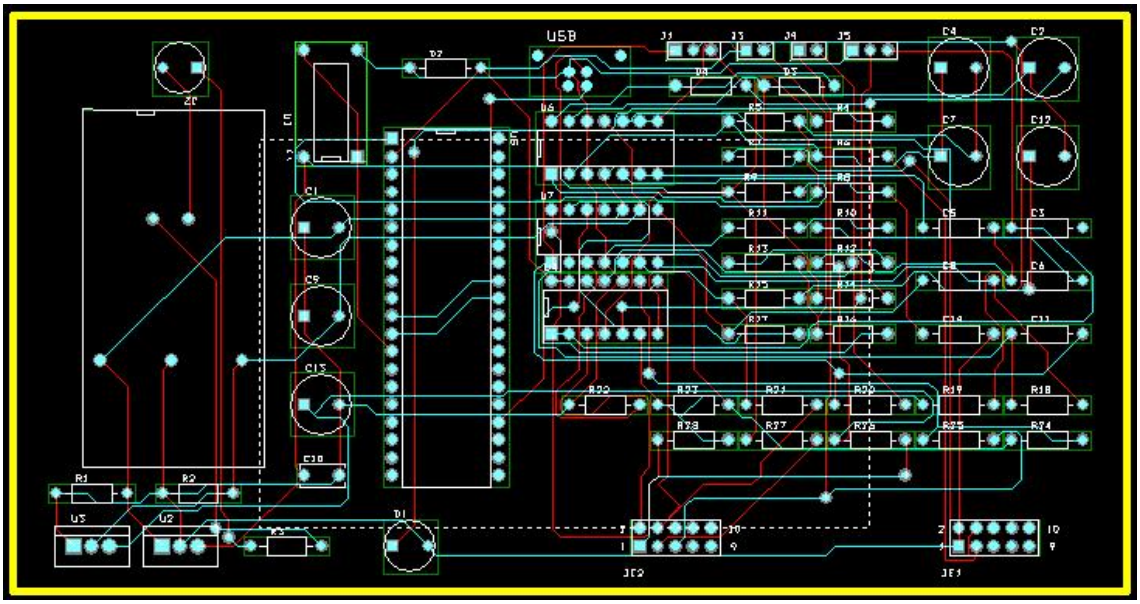


Figura 44 Primer versión funcional positivo

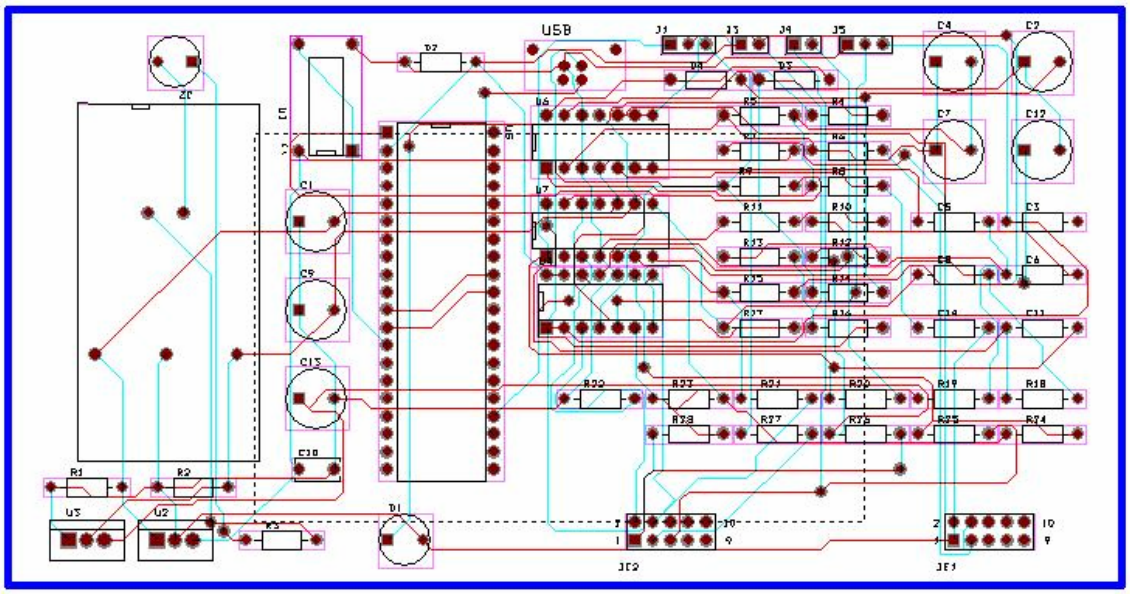


Figura 45 Primer versión funcional negativo

Después de realizar pruebas y reordenar los circuitos a fin de eliminar los componentes no utilizados se obtuvo la segunda versión funcional del sistema, ver figuras 46 y 47, se puede

observar que en esta segunda versión tenemos más espacios libres entre los circuitos, además de que se cambio el circuito de alimentación de los circuitos integrados.

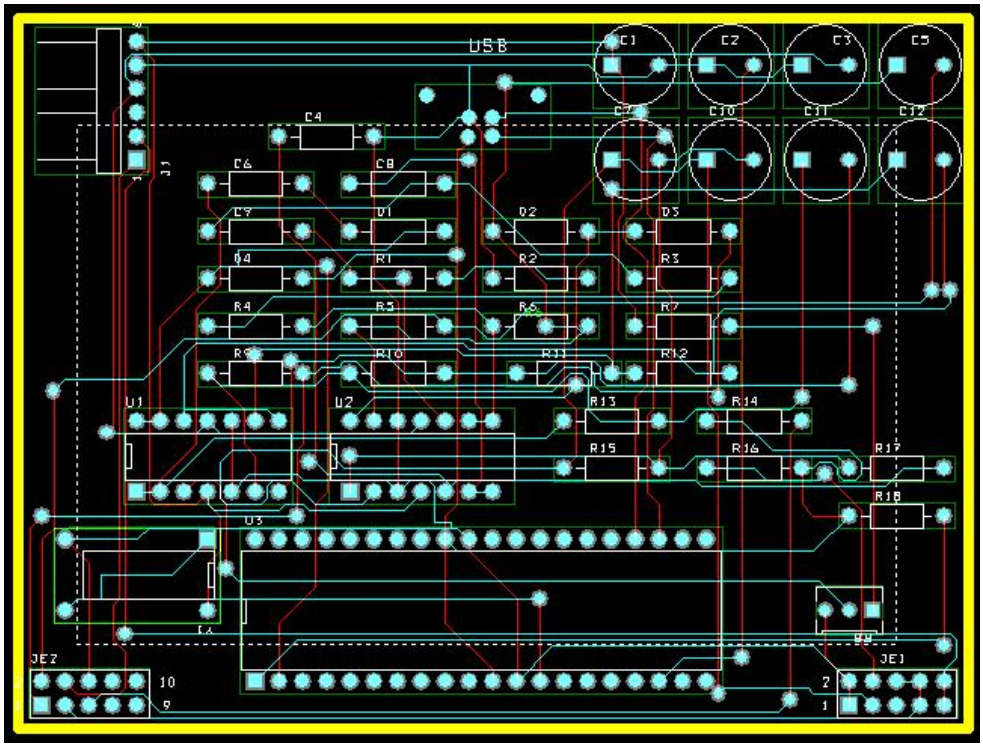


Figura 46 Segunda versión funcional positivo

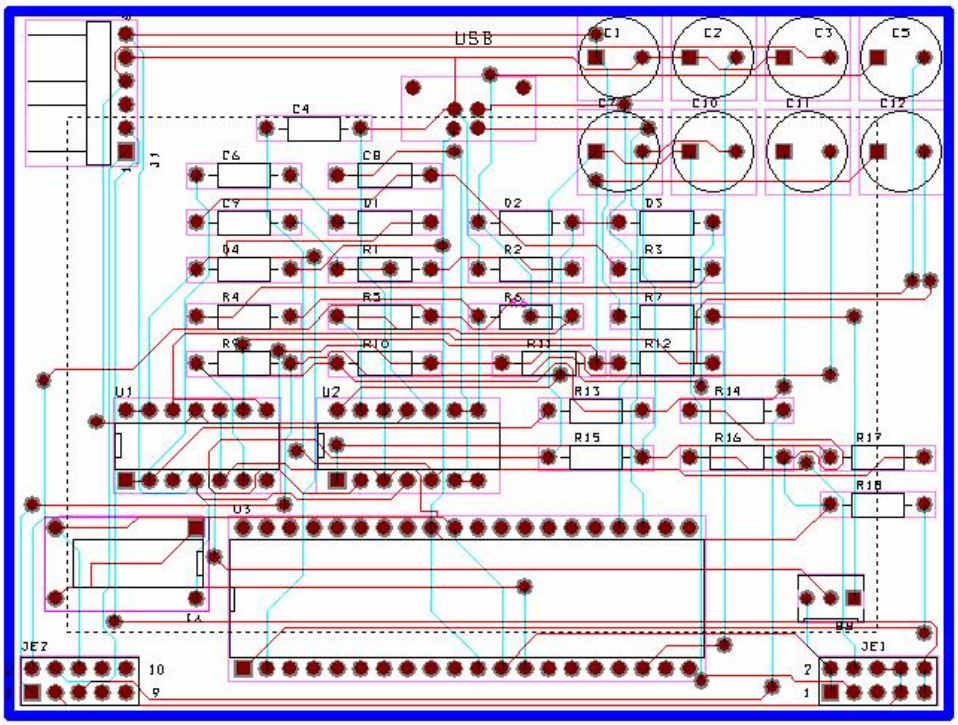


Figura 47 Segunda versión funcional negativo

Finalmente después de algunas modificaciones funcionales y la correcta ocupación de espacios se llego a esta versión, ver figuras 48 y 49, que es considerada la versión final para este trabajo.

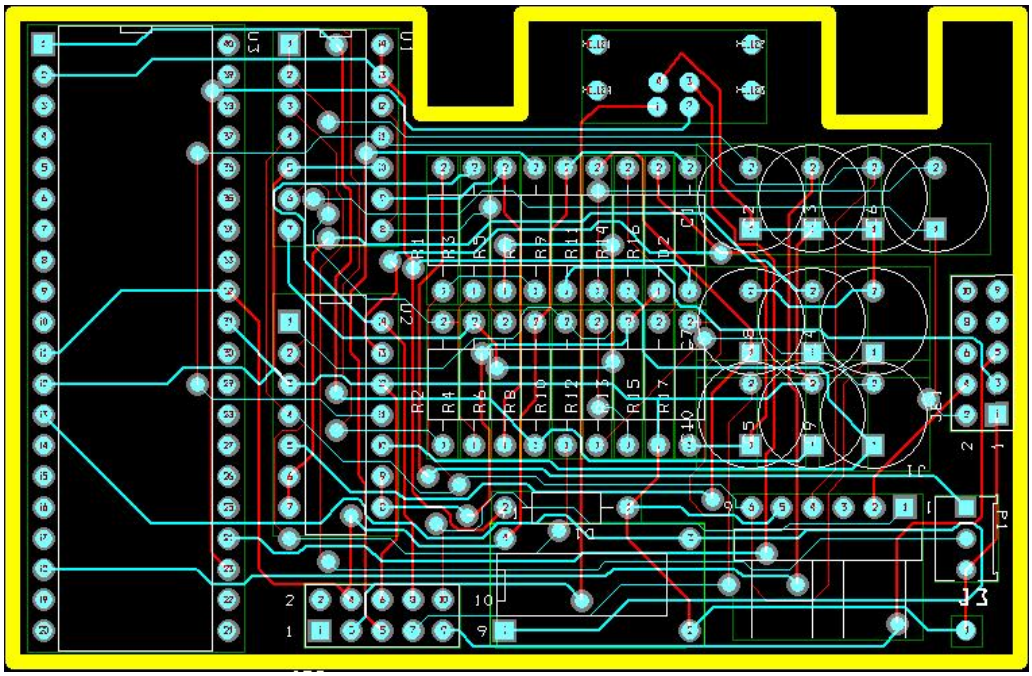


Figura 48 Versión final positivo

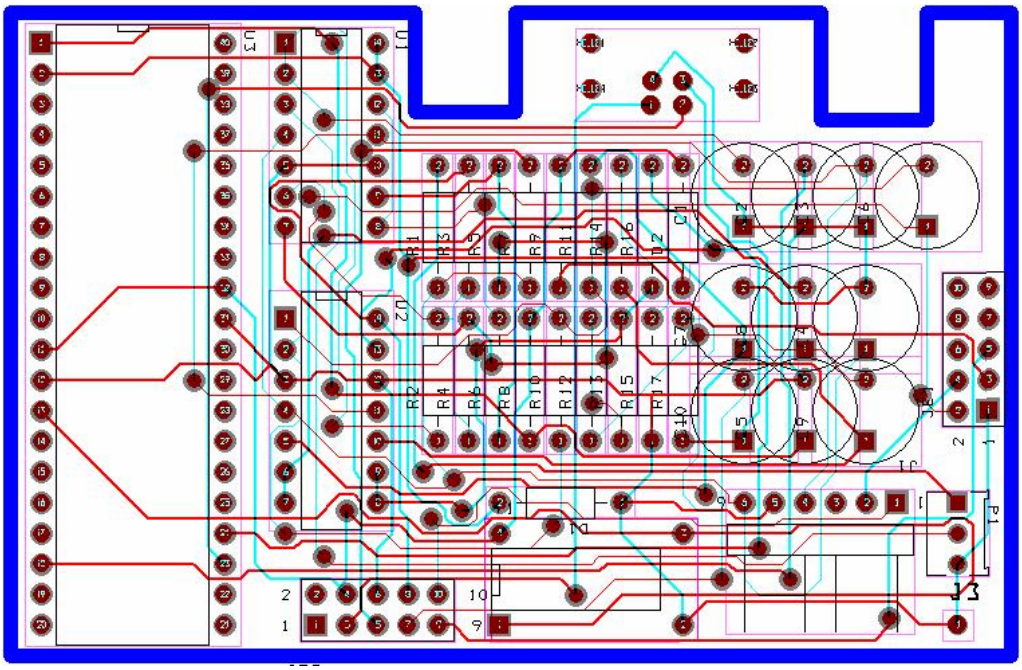


Figura 49 Versión final negativo

El tamaño del circuito impreso para esta versión es el más reducido de todas, lo que reduce el costo de la implementación del mismo.

5.3 El micro controlador

Hasta este momento no se ha explicado con claridad la razón de que se escogiera el MC utilizado en este diseño a continuación se tratara ese tema.

Uno de los principales problemas de los diseños anteriores al propuesto, ver figura 50, estaba relacionado con la velocidad de transmisión de los datos y el problema del almacenamiento de las muestras tomadas por el convertidor analógico digital, dentro del buffer de transmisión de datos.

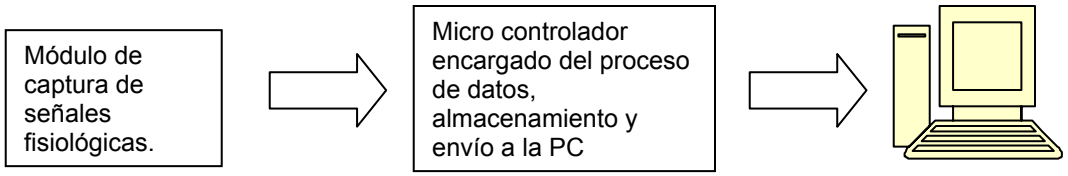


Figura 50 Diagrama de los primeros diseños

Es decir si el puerto USB no estaba listo rápidamente para recibir datos se tenia que esperar a entregar el dato anterior a la PC lo que implicaba la perdida de datos por falta de toma de muestras por parte del MC. La primera idea que se tuvo para corregir este problema fue utilizar dos MC, como se muestra en la figura 51.

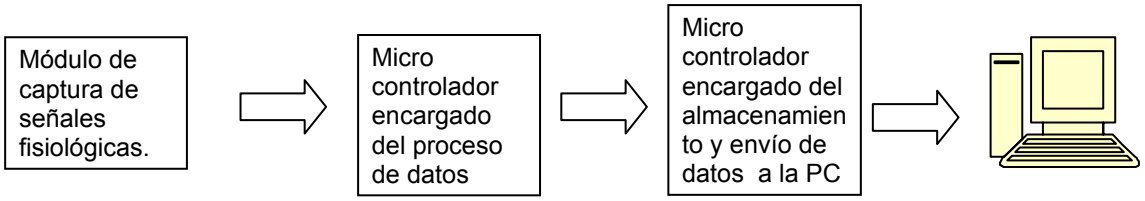


Figura 51 Primera solución

Con esta solución se resolvía de manera efectiva la falla de la perdida de datos por falta de muestreo del MC pero se presentó el problema del desbordamiento de la memoria interna del segundo MC. Por lo que se pensó en utilizar una memoria de almacenamiento intermedio de datos ver figura 52.

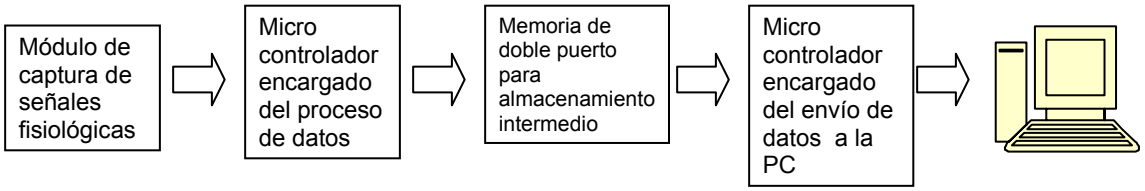


Figura 52 Segunda solución

Esto solucionó el problema del desbordamiento de memoria pero incremento el costo del sistema puesto que las memorias de doble puerto son costosas, por lo que se decidió bajar la velocidad de la toma de datos, esto no afecta el rendimiento global del sistema por que la velocidad de las señales fisiológicas no es muy alta y permitía reducir el tamaño de la memoria de doble puerto tal y como se muestra en la figura 53.

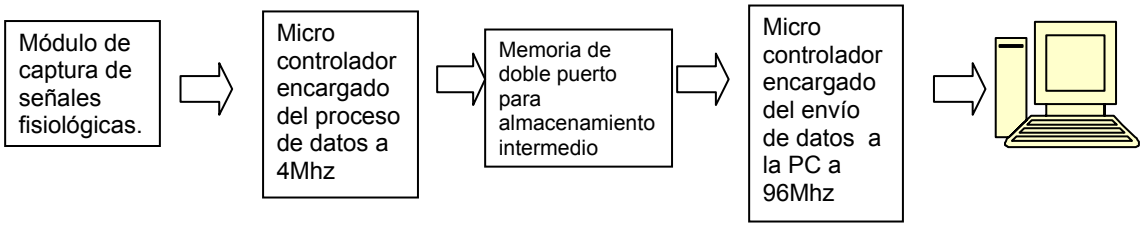


Figura 53 Tercera solución

Esta solución es la que mejor respondía la las necesidades del proyecto por lo que se busco implantarla en un solo bloque cosa que permite el MC utilizado, pues tiene el módulo USB separado del núcleo del MC y utiliza una memoria de doble puerto como vinculo entre ambos módulos, ver figura 54.

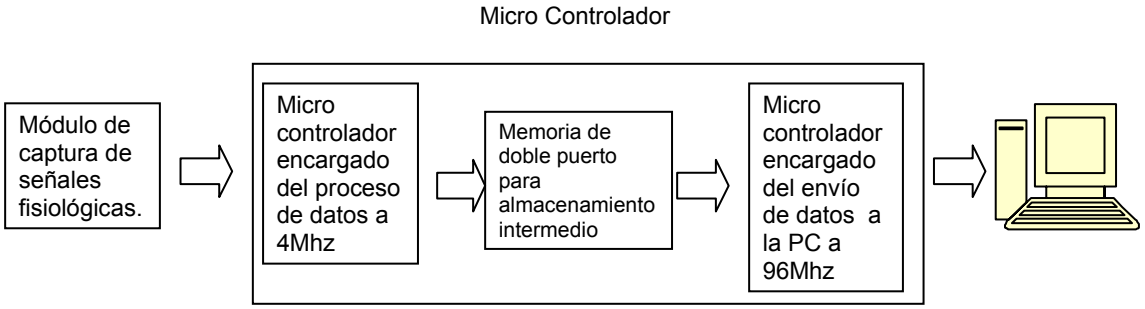


Figura 54 Solución final.

5.4 Software de usuario

Con respecto al bloque del *software* de usuario lo podemos desglosar como se muestra en la figura 55.

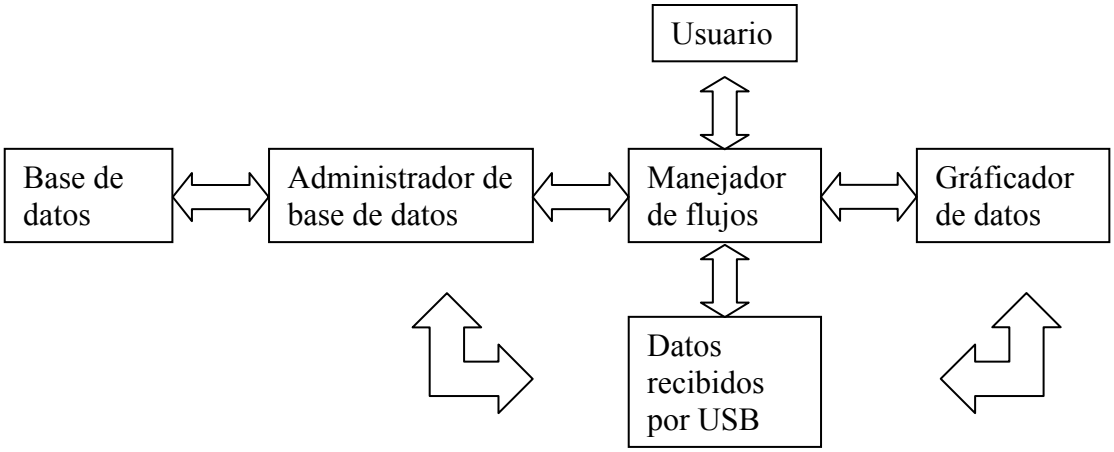


Figura 55 Bloque software de usuario.

Donde:

- **Usuario** es la persona encargada de interactuar con el *software* administra pacientes e interpreta los resultados tanto gráficos como numéricos mostrados por el *software*.

- **Base de datos** es la estructura encargada de almacenar los datos referentes a todas y cada una de las mediciones fisiológicas tomadas a cada paciente, recibe consultas del administrador de base de datos.

- **Administrador de base de datos** es el encargado de escribir y verificar la coherencia de los datos almacenados en la base de datos, así como de formular las consultas a la base de datos.

- **Manejador de flujos** es la máquina de estados que limita las acciones y alcances de los botones u opciones mostradas por el *software,* controla el flujo de toda la información recibida, almacenada y procesada.

- **Gráficador de datos** es el encargado de hacer la representación grafica en pantalla de los datos recibidos a través del puerto USB.

- **Datos recibidos por USB** son los datos capturados por el módulo de adquisición de señales fisiológicas.

6 Conclusiones y trabajo a futuro

6.1 Conclusiones

De este trabajo podemos concluir lo siguiente:

- El sistema es capaz de enviar datos utilizando el protocolo USB 2.0 sin perdida de los mismos.

- No es necesario el uso de *drivers* o librerías para que el *hardware* sea reconocido por el sistema operativo, solo es necesario un archivo de información de la instalación (.inf), que informa al sistema operativo que el dispositivo es compatible con la especificación del archivo usbser.sys.

- El sistema simula que la conexión utilizada no es un USB si no un COM, lo que facilita la programación.

- La programación de nuevas aplicaciones no esta limitada a un solo lenguaje de programación, puesto que el dispositivo se reconoce como una conexión COM.

- No es necesario dar mantenimiento a *drivers* o librerías.

- La resolución o número de muestras por segundo, es de más del doble que la mejor tarjeta existente en el mercado actualmente.

- Se pueden conectar hasta 8 dispositivos de captura de señales fisiológicas sin necesidad de hacer modificaciones al FW de MC.

- Se puede aumentar la velocidad de envío de datos a la computadora puesto que solo se esta utilizando el 60% de la máxima velocidad de transferencia.

- El sistema no se considera invasivo puesto que solo se le conecta un sensor infrarrojo al paciente en alguno de los dedos de la mano.

- El *software* de usuario es muy amigable.

- El *software* de usuario cuenta con un instalador lo cual facilita la distribución del mismo a cualquier usuario final.

- La tarjeta de captura de señales fisiológicas es reproducible en un 100% con funcionamiento idéntico en cada una de ellas.

- Es posible almacenar infinidad de señales adquiridas de un paciente específico sin perdida y con la seguridad de la recuperación de las mismas para su análisis.

- Se cuenta con distintos manuales los cuales simplifican el manejo y mantenimiento del proyecto en general.

- El costo de la tarjeta de captura y envió de señales fisiológicas es, por mucho, menor al costo de la tarjeta de captura comercial más barata del mercado.

A continuación se muestran los resultados del funcionamiento del transmisor de datos USB.

En la siguiente imagen se muestra a la aplicación HyperTerminal de Windows® recibiendo datos del dispositivo de adquisición de señales fisiológicas, no es necesario hacer programación pues esta aplicación puede recibir datos directamente de un puerto COM, con esto se demuestra que la emulación del puerto COM es transparente al sistema operativo.

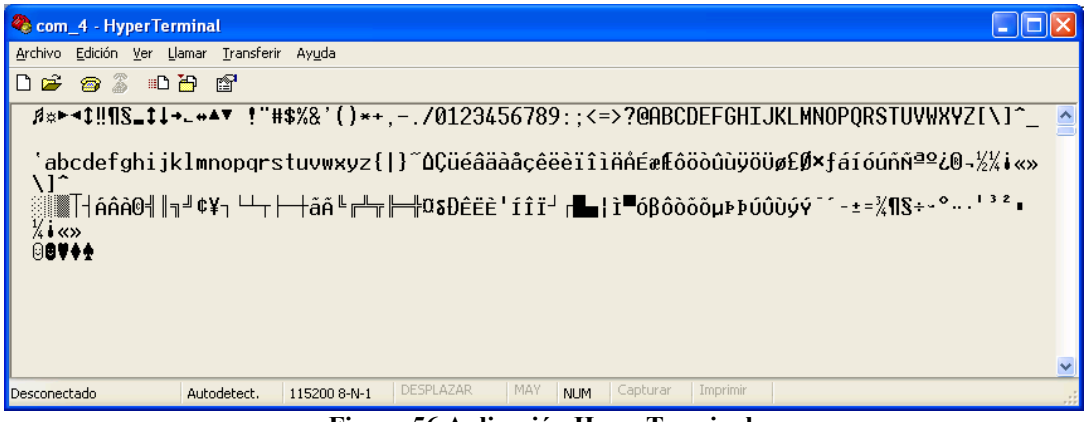


Figura 56 Aplicación HyperTerminal.

En la figura 57 podemos observar cual es la velocidad real de transferencia de datos del dispositivo a la computadora.

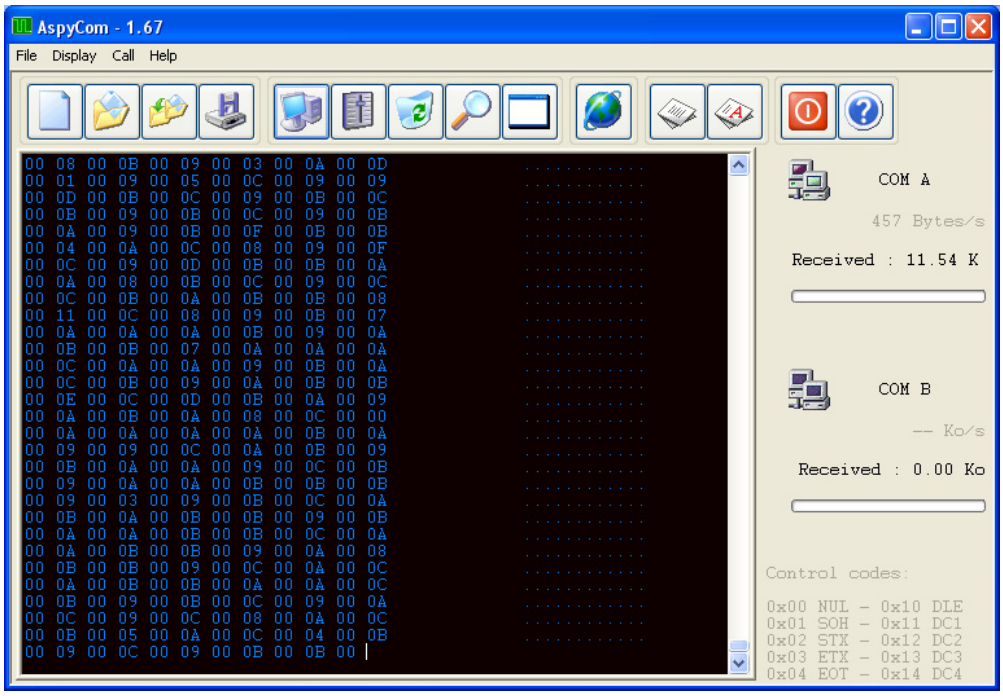


Figura 57 Velocidad de transmisión.

La velocidad real de datos es de 457 Bytes/s, es decir, se reciben 457 puntos por segundo del dispositivo de captura de señales fisiológicas. Esta velocidad no es la máxima alcanzada por el dispositivo de envió USB 2.0 puesto que en el `main ()` del FW existe un lazo de retardo de la señal, el cual limita la velocidad de transmisión a un 60%.
Es decir que la máxima velocidad de envío de datos sin perdida de los mismos por parte del dispositivo USB es de 762 Bytes/s no fue necesario utilizar la máxima velocidad de transmisión puesto que con el 60% de la máxima velocidad se obtenía una resolución 2 veces mayor a la de la mejor tarjeta de captura de señales fisiológicas comercial.

6.2 Trabajo a futuro

Como trabajo a futuro se podría eliminar el uso de alimentación externa, eliminador de voltaje, en la tarjeta de captura de señales fisiológicas e intentar reducir las dimensiones del circuito impreso y por consiguiente el costo del sistema.

En el circuito impreso mostrado a continuación se puede observar un posible diseño de este sistema donde el voltaje se obtiene del puerto USB y se han eliminado algunos componentes no importantes del diseño.

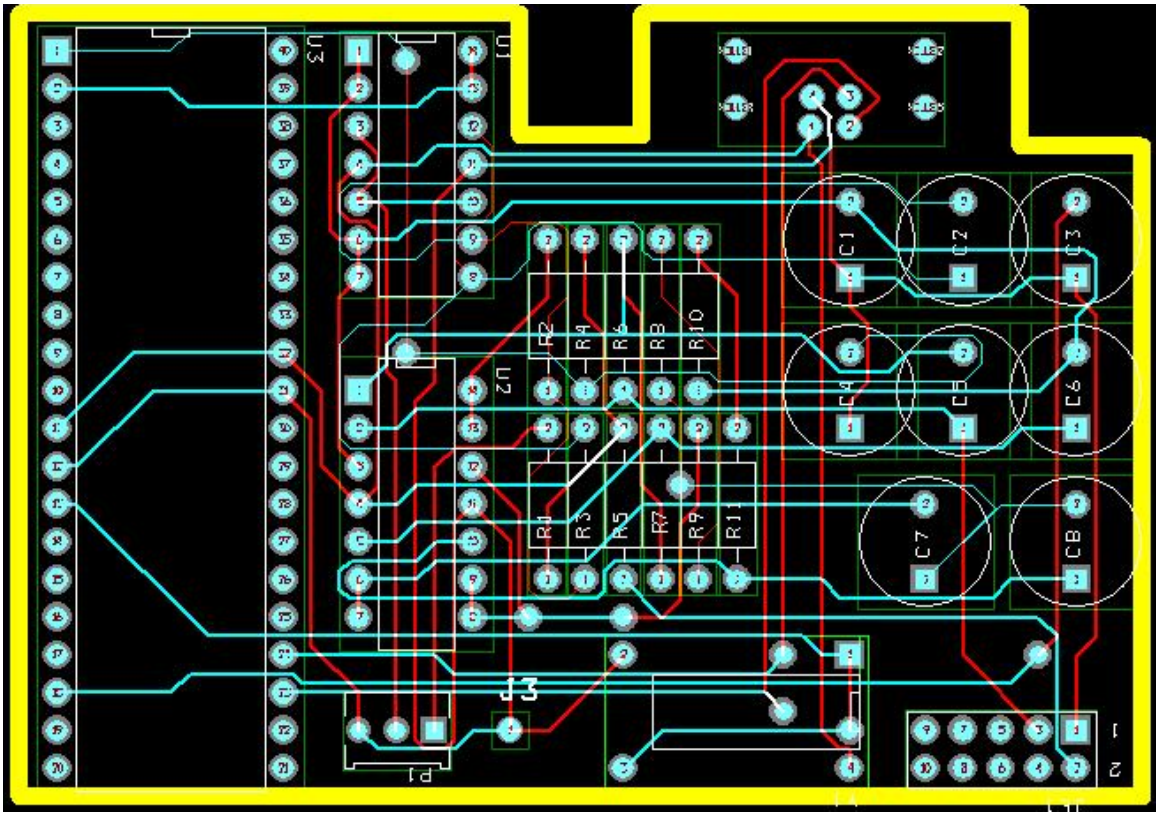


Figura 58 Circuito impreso propuesto.

7 Bibliografía

[1]. La tensión en la vida. Hans Selye. 1956.

[2]. The Use of Fast Fourier Transform for the Estimation of Power Spectra: A Method Based on Time Averaging Overshot modified Periodograms. P.D. Welch. IEEE Trans Audio Electroacustics. 1967.

[3]. A Real – Time QRS Detection Algorithm Jiapu Pan and Willis J. Tompkins, senior member. IEEE, IEEE Transactions on Biomedical Engineering, Vol. BME-32, No. 3 March 1985.

[4]. UNIVERSAL ACTIVE FILTER UAF42, Texas Instruments, http://www.ti.com, 1990.

[5]. Heart Rate Variability. Standards of Measurement, Physiological Interpretation and Clinical Use. Task Force of the European Society of Cardiology and the North American Society of Pacing and Electro physiology. 1996.

[6]. Sistema de Análisis de la Variabilidad de la Frecuencia Cardiaca Grupo de Biología Teórica, Instituto de Investigaciones Biomédicas, UNAM. 1997.

[7]. Circuitos eléctricos, James W. Nilsson : Iowa State University - Susan A. Riedel , Editorial Prentice Hall, 1997

[8]. Java 2 Curso de Programación, Francisco. Javier Ceballos, Editorial Alfaomega Ra-Ma. 2000.

[9]. PICmicro® 18C MCU Family Reference Manual, http://www.microchip.com. Microchip Technology Inc, 2000.

[10]. Universal Serial Bus Specification, Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips, Revision 2.0 April 27, http://www.usb.org 2000.

[11]. Design and evaluation of artifact-resistant finger-ring plethysmographic sensors, Sokwoo Rhee, Boo-Ho Yang, and Haruhiko H. Asada d'Arbeloff, Laboratory for Information Systems and Technology, Department of Mechanical Engineering Massachusetts Institute of Technology Cambridge, MA 02139, U.S.A. 2000.

[12]. Análisis de la variabilidad de intervalos de tiempo del ECG. Alvarado Serrano, Carlos. UPC. ISBN B-4723-2003/84-688-0733-8. Noviembre 2001.

[13]. Circuitos electrónicos con amplificadores operacionales, González de la Rosa, Editorial Marcombo, 2002.

[14]. Piensa en Java, Bruce Eckel, Editorial Prentice Hall. 2002.

[15]. Amplificadores operacionales y circuitos integrados lineales, James M. Fiore, Editorial Agapea, 2002.

[16]. Migrating Applications to USB from RS-232 UART with Minimal Impact on PC Software Rawin Rojvanit, Microchip Technology Inc. http://www.microchip.com. 2002.

[17]. Software de control de fotopletismografo digital, Alexander Pascau, Oreste Ferrer, Manuel Cuadra, Alexis Corzo, Domingo Hernández, Juan Carlos. García, Alfredo Aldama. Centro de Biofísica Médica, Universidad de Oriente 2002.

[18]. TIAPE: Test integral de Auto-Percepción del Estrés. Dra. María Elena Sánchez, Dr. Adriano de Luca Pennacchia. Departamento de Psicología, Universidad Autónoma Metropolitana. Sección de Computación, CINVESTAV. 2003.

[19]. Mobile Monitoring with Wearable Photoplethysmographic Biosensors, H. Harry Asada, Phillip Shaltis, Andrew Reisner, Sokwoo Rhee, and Reginald C. Hutchinson, IEEE Engineering in medicine and biology magazine, 2003.

[20]. Physionet: The research resource for complex physiologic signals. MEMSE, estimate power spectrum using maximum entropy (all poles) method. http://www.physionet.org/. 2004.

[21]. Medidor Multidimensional de Stress. Gregorio Pérez Olán. Cinvestav 2004.

[22]. PIC18F2455/2550/4455/4550 Data Sheet http://www.microchip.com. Microchip Technology Inc, 2004.

[23]. Tips ‘n Tricks Power Manager PICmicro® Featuring nanoWatt Technology http://www.microchip.com. Microchip Technology Inc, 2004.

[24]. An Introduction to USB Descriptors with a Game Port to USB Game Pad Translator Example, Reston Condit, Microchip Technology Inc. http://www.microchip.com. 2004.

[25]. Implementing a PID Controller Using a PIC18 MCU, Chris Valenti, Microchip Technology Inc. http://www.microchip.com. 2004.

[26]. Power Management for PIC18 USB Microcontrollers with nanoWatt Technology Rawin Rojvanit, Microchip Technology Inc. http://www.microchip.com. 2004.

[27]. J-FET input Operational Amplifiers TL08XXX, Texas Instruments, http://www.ti.com 2004.

[28]. Manejo Estructurado y Análisis de la Información del Medidor Multidimensional de Stress. Enrique Bonilla Henríquez. Cinvestav 2005.

[29]. Flash Microcontroller Programming Specification, http://www.microchip.com. Microchip Technology Inc, 2005.

[30]. Control de un módulo bluetooth mediante microcontrolador, Jaime Gálvez Navarro, Universidad Politécnica de Cataluña, 2005

[31]. Micro-circulation of skin blood: optical monitoring by advanced photoplethysmography techniques, Janis Spigulis*, Renars Erts and Uldis Rubins, University of Latvia, Physics Department and IAPS, Raina Blvd. 19, Riga, LV-1586, Latvia 2005.

[32]. The American Institute of Stress. America's #1 Health Problem. http://www.stress.org. 2005.

[33]. Medidores portátiles de ECG www.nymedequip.com, 2005.

[34]. Proyectos y usos de circuitos electrónicos http://electronica.eia.edu.co/, 2005.

[35]. Función pulmonar y su evaluación http://www.uninet.edu/, 2005.

[36]. Biopotential Technology http://openeeg.sourceforge.net/doc/links-biopsy.html, 2005.

[37]. Electrocardiogram http://www.cs.wright.edu/, 2005.

[38]. Engineering in Medicine and Biology Society. http://www.eng.unsw.edu.au/embs/, 2005.

[39]. Power SMD LED Datasheet TLMK 3302, Vishay, http://www.vishay.com/docs/83201/83201.pdf. 2006.

[40]. Photodiode silicon Datasheet BPW 34 Osram. http://www.osram.com. 2006.

[41]. Quad operational amplifier LM224N Datasheet, Fairchild, http://www.fairchild.com, 2006.

[42]. Universal Serial Bus Specification, Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, Philips, Revision 2.1 http://www.usb.org , 2006.

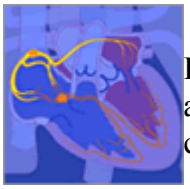
8 Apéndice

A Señales eléctricas y flujo sanguíneo

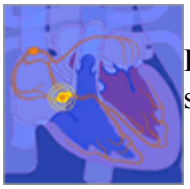
Las señales eléctricas creadas por el nodo S-A siguen una vía eléctrica natural atravesando las paredes del corazón. El movimiento de las señales eléctricas hace que las cavidades del corazón se contraigan y se relajen. Cuando una señal pasa por la pared de una cavidad, esta se contrae. Cuando la señal abandona la pared, la cavidad se relaja. En un corazón sano, las cavidades se contraen y relajan de una forma coordinada o siguiendo un *ritmo.*
Cuando el corazón late con un ritmo a una frecuencia normal, hablamos de un *ritmo sinusal.* Una alteración en el sistema eléctrico del corazón puede alterar el ritmo cardiaco normal. Cualquier tipo de ritmo o frecuencia cardiaca anormal se denomina *arritmia.* Es normal y saludable que su latido se acelere o retarde a lo largo del día en función de los cambios en el nivel de actividad. Sin embargo, no es normal que el corazón lata fuera de su ritmo. Si el corazón late fuera de su ritmo, no puede suministrar sangre suficiente al organismo.

A.1 El camino que siguen las señales eléctricas

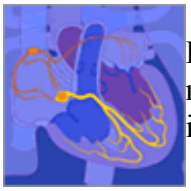
PASO 1. El nodo S-A (marcapasos natural) genera una señal eléctrica.



PASO 2. La señal eléctrica sigue las vías eléctricas naturales pasando por ambas aurículas. El movimiento de la electricidad hace que las aurículas se contraigan ayudando a impulsar la sangre hacia los ventrículos.



PASO 3. La señal eléctrica llega al nodo A-V (puente eléctrico). Allí, la señal se detiene para dar tiempo a los ventrículos a llenarse con sangre



PASO 4. La señal eléctrica se propaga por el sistema His-Purkinje. El movimiento de la electricidad hace que los ventrículos se contraigan e impulsen la sangre hacia los pulmones y el cuerpo.

A.2 Registro de la actividad eléctrica del corazón, el electrocardiograma (ECG)

Un ECG es un registro de la actividad eléctrica del corazón y se obtiene conectando a un paciente un aparato transductor, que convierte los impulsos eléctricos del corazón en señales digitales que se pueden mostrar en una pantalla para poder ser analizados.

A.2 Partes de un ECG

Las crestas y valles en un registro de ECG se denominan ondas. Cada onda muestra al médico la historia del trabajo que está realizando el corazón.

- La *onda P* muestra las contracciones de las cavidades superiores del corazón (aurículas)
- El *complejo QRS* muestra las contracciones de las cavidades inferiores del corazón (ventrículos)
- La *onda T* muestra cómo se relajan las cavidades inferiores del corazón (ventrículos)

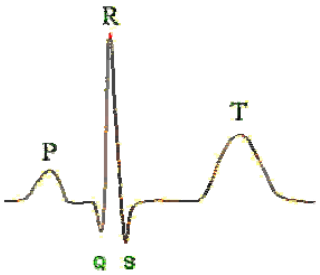


Figura 59 Partes de un ECG.

El análisis de estos valores o de las distancias entre valores del mismo tipo R-R brindan información relacionada con el estado de salud del corazón.

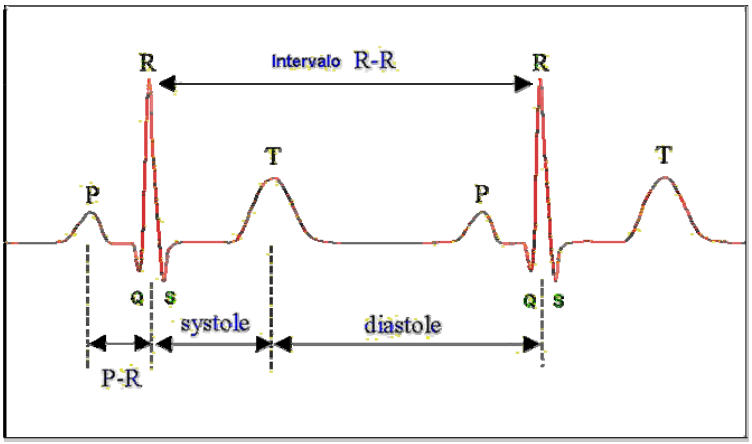


Figura 60 Intervalo R-R

B Introducción a los amplificadores operacionales

Un amplificador operacional es un circuito electrónico (normalmente se presenta como circuito integrado) que tiene dos entradas y una salida. La salida es la diferencia de las dos entradas multiplicada por un factor (G):

$$Vout = G \cdot (V_+ - V_-)$$

Originalmente los amplificadores operacionales se empleaban para operaciones matemáticas (suma, resta, multiplicación, división, integración, derivación, etc) en calculadoras analógicas. De ahí su nombre.
El amplificador operacional ideal tiene una ganancia infinita, una impedancia de entrada infinita, un ancho de banda también infinito, una impedancia de salida nula y ningún ruido. Como la impedancia de entrada es infinita también se dice que las corrientes de entrada son cero.

El símbolo de un amplificador operacional es el mostrado en la figura 61:

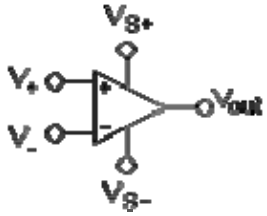


Figura 61 Símbolo del amplificador operacional.

Los terminales son:

- V_+: entrada no inversora
- V_-: entrada inversora
- V_{OUT}: salida
- V_{S+}: alimentación positiva
- V_{S-}: alimentación negativa

Los pines de alimentación pueden recibir diferentes nombres, por ejemplo en los amplificadores operacionales basados en FET V_{DD} y V_{SS} respectivamente. Para los basados en BJT son V_{CC} y V_{EE}. En este proyecto se utilizaron varias configuraciones conocidas de los amplificadores operacionales entre las cuales están.

- Comparador

Figura 62 Comparador de señal.

Esta es una aplicación sin realimentación. Compara entre las dos entradas y saca una salida en función de qué entrada sea mayor. Se puede usar para adaptar niveles lógicos.

$$V_{out} = \begin{cases} V_{S+} & V_1 > V_2 \\ V_{S-} & V_1 < V_2 \end{cases}$$

- Seguidor

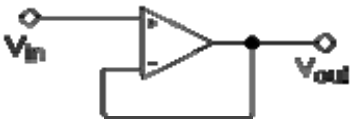


Figura 63 Seguidor de señal.

Se usa como un buffer, para eliminar efectos de carga o para adaptar impedancias (conectar un dispositivo con gran impedancia a otro con baja impedancia y viceversa)
Como la tensión en las dos patillas de entradas es igual: $V_{out} = V_{in}$

$$Z_{in} = \infty$$

- Inversor

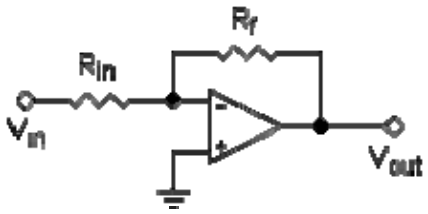


Figura 64 Inversor de señal.

El análisis de este circuito es el siguiente:

$$V_+ = V_- = 0$$

Definiendo corrientes: $\frac{V_{in} - 0}{R_{in}} = -\frac{V_{out} - 0}{R_f}$ y de aquí se despeja

$$V_{OUT} = -V_{in}\frac{R_f}{R_{in}}$$

Se observa que la salida resultante es el negativo de la señal de entrada multiplicada por un factor. Para el resto de circuitos el análisis es similar.

$Z_{in} = R_{in}$

- No inversor

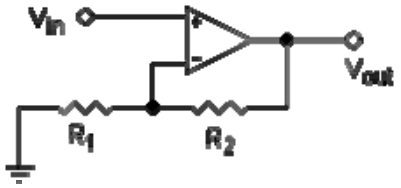


Figura 65 No inversor de señal.

$$V_{out} = V_{in}(1 + \frac{R_2}{R_1})$$

$Z_{in} = \infty$

- Integrador

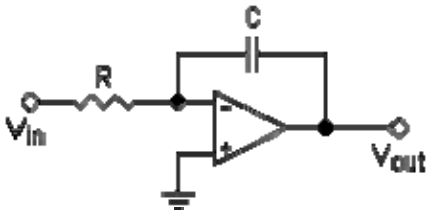


Figura 66 Integrador de señal.

Integra e invierte la señal (V_{in} y V_{out} son funciones dependientes del tiempo)

$$V_{out} = \int_0^t -\frac{V_{in}}{RC}\,dt + V_{inicial}$$

$V_{inicial}$ es la tensión de salida en el origen de tiempos (t = 0). Este circuito también se usa como filtro

- Derivador

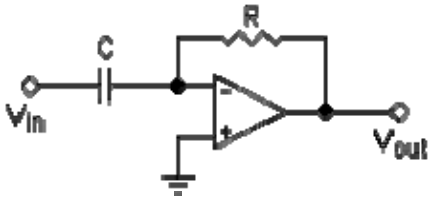


Figura 67 Derivador de señal.

Deriva e invierte la señal respecto al tiempo

$$V_{out} = -RC\,\frac{dV_{in}}{dt}$$

Este circuito también se usa como filtro

C Descripción del protocolo USB 2.0 y diferencias respecto a la Versión 1.0

C.1 Tramas y Microtramas

USB 1.x dividía el tiempo en tramas de 1 milisegundo. Adicionalmente, USB 2.0 define un tiempo de microtrama de 125 microsegundos.
Al igual que en USB 1.x se reservan ciertos porcentajes del tiempo de trama, para dar servicio a las distintas transacciones de Control, Interrupción e Isócronas *full/low-speed*, en USB 2.0 también se reservan ciertos porcentajes del tiempo de microtrama, para dar servicio a los distintos tipos de transacciones de alta velocidad.

Endpoints High-Speed y *High Bandwidth*

En USB 1.x, el tiempo máximo reservado en cada trama a cada *endpoint* Isócrono o de Interrupción es el de una transacción por trama.

USB 2.0 soporta dos tipos de *endpoints* Isócronos y de Interrupción:

Endpoints de ancho de banda normal, que precisan hasta 1,024 bytes por microtrama (una transacción).

Endpoints de alto ancho de banda (*High Bandwidth*), que precisan más de 1,024 bytes por microtrama, hasta un máximo de 3,072 bytes (3 transacciones).

C.2 Protocolo I: paquetes y transacciones

USB 2.0 mantiene la arquitectura centralizada definida en USB 1.x, en la que el *host* es el iniciador de todas las transferencias que se producen en el bus. Los dispositivos deben mantenerse a la espera hasta recibir del *host* un paquete especial (*token*) dirigido a él, indicando el tipo de transferencia a realizar.
USB 2.0 también mantiene la estructura de la transacción, formada en base a la secuencia de paquetes *Token*-Dato-Validación (*Handshake*), definida en USB 1.x. Básicamente USB 2.0 añade algunos tipos de paquetes nuevos, para implementar las nuevas funciones y protocolos que se han incorporado.

En el grupo de paquetes de *Token*, siguen existiendo los definidos en USB 1.x (*IN* (direcciones de dispositivo transmisor de los datos), *OUT* (direcciones de dispositivo destinatario de los datos), SOF (indicador de comienzo y numeración de trama) y SETUP (direcciones de dispositivo destinatario de comandos de control)), aunque USB 2.0 reutiliza el tipo SOF:

USB 1.x utiliza el *token* SOF para indicar el principio de trama (una vez cada milisegundo). USB 2.0 usa este mismo *token* para indicar el principio de microtrama (una vez cada 125 microsegundos).

En el grupo de paquetes de Datos, aparte de los dos tipos definidos en USB 1.x (DATA0 (paquete de datos par) y DATA1 (paquete de datos impar)), USB 2.0 define dos nuevos tipos:

- DATA2, empleado en transacciones isócronas de alta velocidad y alto ancho de banda.
- MDATA, empleado en transacciones isócronas de alta velocidad y alto ancho de banda y en transacciones *split* (las transacciones se describen más adelante).

En el grupo de paquetes de Validación (h*andshake*), aparte de los tres tipos definidos en USB 1.x (ACK (recepción de paquete de datos libre de errores), NAK (receptor no puede aceptar datos o el transmisor no puede enviar) y STALL (los dispositivos TX/RX detenidos o no pueden soportar un comando)), USB 2.0 define un nuevo paquete:

NYET, significa "*Not YET*" (todavía no), y se emplea en los protocolos *split* y de control de flujo PING.

Por último, en el grupo de paquetes especial, USB 2.0 define tres nuevos tipos de paquetes, reutilizando uno de ellos (ERR) el código asignado por USB 1.x al paquete especial PRE. Se recuerda aquí la descripción del paquete PRE, para mostrar que no hay posibilidad de error en cuanto a que un mismo código identifique a dos paquetes distintos.

PRE (Preámbulo): Definido en USB 1.x para indicar que a continuación se va a transmitir en modo baja velocidad. Sólo lo puede transmitir un *host* USB 1.x (en modo de velocidad total) inmediatamente antes del campo de *token* de la transacción de baja velocidad.

ERR (Error): Se usa para indicar errores en transacciones *Split high-speed*. Reutiliza el código del tipo de paquete PRE. Sólo lo puede transmitir un concentrador USB 2.0 (en modo *high-speed*) en el campo de validación (*handshake*) de una transacción s*plit*.

SPLIT: Lo transmite el *host* en el campo de *token* de una transacción s*plit*.

PING: Lo transmite el *host* en el campo de *token* de una transacción de control de flujo PING.

C.3 Paquete de token SOF y número de microtrama

En USB 1.x, el número de la trama es un campo de 11 bits que el *host* incrementa una vez por trama, y que envía sólo en el paquete SOF (*Start-Of-Frame*) al principio de cada trama. USB 2.0 utiliza este mismo campo para indicar el número de microtrama. En este caso, el

host no envía todos los bits del número de microtrama (14 bits), sino solamente los 11 bits más altos. Como cada trama (1 mili segundos) contiene 8 microtramas (125 micro segundos), el efecto de lo anterior es que el *host* USB 2.0 envía durante 8 microtramas consecutivas un paquete SOF con el mismo número de trama. Los dispositivos *full-speed* reciben un paquete SOF cada milisegundo con un número de trama secuencial. Los dispositivos *high-speed* reciben un paquete SOF cada 125 microsegundos, pero con el mismo número de trama en cada grupo de 8 consecutivos.
Los dispositivos *high-speed* que necesiten llevar un control del número de microtrama, pueden detectar cuándo un paquete SOF tiene un número de trama distinto del anterior, e iniciar un contador interno de microtrama (de 3 bits) a 0. Cada uno de los siguientes 7 paquetes SOF con el mismo número de trama serán los correspondientes a las microtramas 1 a 7.

C.4 Paquetes de datos

USB 2.0 sigue haciendo un uso de los paquetes DATA0 y DATA1, como parte del mecanismo de detección de errores *Data Toggle* en los casos definidos en USB 1.x; adicionalmente USB 2.0 define los nuevos paquetes DATA2 y MDATA, de nuevo con la función de detección de errores en los nuevos casos definidos en USB 2.0.

El uso de los paquetes de datos en los nuevos casos definidos en USB 2.0 es:

- Las transacciones de Interrupción *high-speed high bandwidth* realizadas en la misma microtrama, hacen uso del mecanismo *Data Toggle* (uso alternativo de paquetes DATA0 y DATA1).

- En transacciones Isócronas-IN *high-speed high-bandwidth*, el dispositivo envía los datos utilizando las siguientes secuencias de paquetes:

    1. En el caso de 1 transacción por microtrama, se usa el paquete DATA0.
    2. En el caso de 2 transacciones por microtrama, se usa la secuencia de paquetes DATA1-DATA0.
    3. En el caso de 3 transacciones por microtrama, se usa la secuencia de paquetes DATA2-DATA1-DATA0.

- En transacciones Isócronas-OUT *high-speed high-bandwidth*, el *host* envía los datos utilizando las siguientes secuencias de paquetes:

    1. En el caso de 1 transacción por microtrama, se usa el paquete DATA0.
    2. En el caso de 2 transacciones por microtrama, se usa la secuencia de paquetes MDATADATA1.
    3. En el caso de 3 transacciones por microtrama, se usa la secuencia de paquetes MDATA MDATA-DATA2.

- En transacciones *Start-Split* de Interrupción-OUT, el *host* alterna los paquetes DATA0 y DATA1 en la manera habitual (*data toggle*).

- En transacciones *Complete-Split* de Interrupción-IN, el concentrador también alterna los paquetes DATA0 y DATA1, siempre que la transacción en el bus *full/low-speed* se haya completado durante una microtrama. Cuando se da el caso de que la transacción en el bus *full/low-speed* abarca dos microtramas, la respuesta del concentrador en la primera transacción *Complete-Split* usa el paquete MDATA, para indicar al *host* que faltan datos por enviar, y que debe ejecutar otra transacción *Complete-Split* en la siguiente microtrama, para obtener el resto de los datos (que se transmitirán mediante un paquete DATA0 o DATA1, según corresponda en la secuencia *Data Toggle*).

- En transacciones *Start-Split Isócronas-OUT*, el host siempre usa el paquete DATA0.

- En transacciones *Complete-Split Isócronas-IN*, el concentrador usa el paquete MDATA para todos los paquetes excepto para el último, en que usa el paquete DATA0 (la transacción *full-speed* puede abarcar varias microtramas, por lo que los paquetes MDATA informan al host que debe seguir enviando transacciones *Complete-Split* en cada microtrama, hasta que reciba un paquete DATA0).

Nota: La máxima cantidad de información útil que se puede transferir en un paquete de datos depende de la velocidad del dispositivo y del tipo de transferencia.

C.5 Paquete de validación (handshake) NYET

Se utiliza sólo en modo *high-speed*, y se puede transmitir en dos circunstancias:
Lo puede transmitir un *endpoint high-speed* tipo *Bulk-OUT* o de Control, como respuesta durante el protocolo de control de flujo PING.
También lo puede transmitir un concentrador *high-speed* como respuesta a una transacción *Split*, cuando el concentrador todavía no ha concluido la transacción *full/low-speed* con el dispositivo, o bien cuando el concentrador no puede atender la transacción en ese momento.

C.6 Paquete especial de handshake ERR

Se utiliza sólo en modo *high-speed*, y lo puede transmitir un concentrador *high-speed* en el campo de *handshake* de una transacción *Complete-Split*, para informar de un error en la transacción *full/low-speed*.

C.7 Paquetes especiales de token para transacciones split

Aparte de las transacciones definidas en USB 1.x (Control, *Bulk*, Interrupción e Isócronas), USB 2.0 define las transacciones *Split*. Sólo los controladores de *host* y los concentradores

USB 2.0 deben soportar este nuevo tipo de transacción, ya que es invisible a los dispositivos.
Este tipo de transacción es el que permite la comunicación con dispositivos *full/low speed* conectados a concentradores que funcionan en modo *high-speed*. El *host* comienza una transacción *Split* cuando envía al concentrador, en modo *high-speed*, toda la información necesaria para que el concentrador ejecute ahora una transacción *full/low-speed* con el dispositivo. Toda la información queda almacenada en el concentrador, en el TT correspondiente al puerto al que está conectado el dispositivo (o en el único TT disponible si se trata de un concentrador mono-TT). Mientras el concentrador ejecuta esta transacción con el dispositivo *full/low-speed*, el bus queda libre para ejecutar nuevas transacciones *high-speed* con otros dispositivos.
La respuesta del dispositivo *full/low-speed* queda almacenada a su vez en el TT del concentrador, disponible para cuando el *host* posteriormente indique al concentrador que la envíe, en modo *high-speed*.
USB 2.0 define el *token* especial *SPLIT* para llevar a cabo las transacciones *Split*. Este es el único paquete de *token* de 4 bytes, a diferencia de los paquetes de *token* normales de 3 bytes.

USB 2.0 define dos transacciones *Split* que hacen uso del *token* especial SPLIT:

- **Start Split**: La utiliza el *host* para enviar al concentrador, en modo *high-speed*, toda la información necesaria para que el concentrador ejecute ahora una transacción *full/low-speed* con el dispositivo.
- **Complete Split**: La utiliza el *host* para leer del concentrador, en modo *high-speed*, la respuesta del dispositivo *full/low-speed*.

Un campo en el propio paquete SPLIT identifica el tipo de transacción.

C.8 Paquete especial de token PING para control de flujo

Un caso bastante frecuente y que produce una gran pérdida de ancho de banda útil en un bus USB 1.x, se da cuando los dispositivos *full/low-speed* contienen *endpoints* tipo *Bulk*-OUT y de Control que necesitan un tiempo para procesar los datos recibidos, de forma que no pueden momentáneamente recibir nuevos datos hasta que se desocupe el buffer de recepción.
En USB 1.x, esta situación la controla el dispositivo devolviendo una validación (*handshake*) NAK en la transacción OUT en la que el *host* ha transmitido un nuevo paquete de datos. Esta validación indica que el dispositivo no ha podido recibir los datos porque no tiene espacio suficiente en su *buffer* de recepción, y el *host* debe reintentar la transmisión posteriormente. Desafortunadamente, para cuando el dispositivo informa que no tiene espacio, la mayor parte del tiempo de la transacción ya se ha consumido, ya que el paquete de datos se ha transferido íntegramente. Esto produce una pobre utilización del bus cuando se suceden múltiples transacciones OUT con respuesta negativa (NAK) por parte del dispositivo.

USB 2.0 define un nuevo protocolo de control de flujo más eficiente, denominado PING, que debe ser utilizado por las transferencias de tipo *Bulk*-OUT y de Control. Las transferencias de Control deben soportar este protocolo en las fases de Datos y de Estado, pero no en la fase de *Setup*. Este nuevo protocolo evita la transmisión de paquetes de datos hasta que el *host* sabe que el dispositivo puede aceptarlos.
El *host* pregunta al dispositivo *high-speed* mediante el paquete especial de *token* PING, y el dispositivo puede contestar con una validación ACK, para indicar que tiene espacio para recibir un nuevo paquete de datos, o con un *handshake* NAK, para indicar que no tiene espacio. El *host* pregunta periódicamente mediante paquetes PING hasta que recibe un *handshake* ACK, en cuyo caso procede a la transmisión del paquete de datos.

Una vez que se produce la recepción de un paquete de datos, el dispositivo puede contestar con una validación ACK, para indicar la correcta recepción del paquete y que tiene espacio para el siguiente paquete, o con una validación NYET, para indicar la correcta recepción del paquete y que no tiene espacio para el siguiente paquete.
El *host* puede seguir transmitiendo paquetes de datos en tanto que el dispositivo siga contestando con ACK.
En el momento en que el dispositivo conteste con NYET, el *host* debe volver al proceso de preguntar al dispositivo mediante paquetes PING antes de enviar un nuevo paquete de datos.

D Protocolo II: tuberías y transferencias

USB 1.x define las vías de comunicación entre las aplicaciones que se ejecutan en el *host* (clientes) y los distintas *endpoints* en los dispositivos USB (servidores), y las denomina *"pipes"* (tuberías). Cuando un dispositivo USB se conecta a un sistema, y el sistema lo reconoce y lo configura, el dispositivo queda organizado como un cierto conjunto de *endpoints* de distintos tipos (existen 4 tipos de *endpoints*). Entonces el sistema establece todas las vías de comunicación (tuberías) necesarias entre el sistema y cada uno de los *endpoints* disponibles en dicha configuración. El dispositivo puede implementar varias posibles configuraciones, con distintos conjuntos de de distintos tipos transferencias en cada una de ellas. El sistema elige una cierta configuración en función de la funcionalidad particular que se precise del dispositivo.

Existen 4 tipos de *endpoints* (*Bulk*, Control, Interrupción e Isócrono) y 2 tipos de tuberías (Control o Mensaje y *Stream*). Las posibles combinaciones son:

1. Tuberia de Control o Mensaje. Es una vía de comunicación bidireccional entre el *host* y dos *endpoints* de Control en un dispositivo USB. Un *endpoint* es de Salida y el otro es de Entrada, de forma que se pueda establecer la comunicación bidireccional. Todos los dispositivos USB disponen de dos *endpoints* de Control en la dirección 0, uno de entrada y uno de salida, de manera que el sistema siempre puede establecer una tubería de Control con el dispositivo, incluso antes de configurarlo (se denomina Tubería de Control por Defecto, y es la única tubería que se puede establecer antes de configurar al dispositivo). A través de esta tubería, el

sistema puede leer del dispositivo toda la información descriptiva necesaria para enterarse del tipo de dispositivo, posibles configuraciones, protocolos que soporta, número y tipos de *endpoints* que soporta en cada posible configuración, etc. Esta información descriptiva son los Descriptores.

2. Tubería *Stream*. Es una vía de comunicación unidireccional entre el *host* y un *endpoint* de los tipos *Bulk*, Interrupción o Isócrono. Si un dispositivo necesita transferencias bidireccionales de un tipo de *endpoint* concreto, el sistema debe establecer dos tuberías, una de salida (con un *endpoint* de salida) y otra de entrada (con un *endpoint* de entrada).

Las diferencias básicas entre las transferencias USB 1.x y USB 2.0 son:
- Tamaños máximos de los paquetes de datos en cada tipo de transferencia.
- Reserva de tiempo de microtrama para las transferencias de Control, de Interrupción e Isócronas.

D.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, ó 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 de trama o microtrama:

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

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

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

- Si el tiempo de trama o microtrama 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 la trama o microtrama 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 una trama o microtrama, el *host* selecciona cuáles se procesan, quedando el resto pendientes para una próxima trama o microtrama.

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

D.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 (isocronismo).

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 trama de hasta 1,023 bytes.
- *High-speed*: 1 transacción por microtrama de hasta 1,024 bytes.
- *High-speed high-bandwidth*: 2 ó 3 transacciones por microtrama de hasta 1,024 bytes cada una.

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 de trama o microtrama para realizarlas. Para ello, los *endpoints* isócronos indican qué cantidad de información como máximo debe transferir la tubería en cada trama o microtrama, 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 de trama o microtrama 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 de trama.
- *High-Speed*: Hasta un 80 % del tiempo de microtrama.

D.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 microtrama de hasta 1,024 bytes cada una.

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 de trama o microtrama 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 us a 4´096 seg.

La reserva de tiempo de trama o microtrama 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 de trama.
- *High-Speed*: Hasta un 80 % del tiempo de microtrama.

D.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 que 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 de los paquetes de datos son:

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

USB hace una gestión "*good effort*" para ir dando curso a 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 una trama o microtrama.
Los *endpoints Bulk-OUT high-speed* soportan el protocolo de control de flujo PING.