Índice de contenido

Índice de tablas

Índice de ilustraciones

CAPÍTULO 1:

1. INTRODUCCIÓN

En este capítulo abordaremos una introducción a la digitalización de señales de pequeña amplitud, para posteriormente explicar una visión general de las características de las señales de electrocardiograma. Posteriormente expondremos cuáles son los problemas en la digitalización de señales de electrocardiograma para dar un enfoque de cual es la constitución del proyecto y una breve reseña de las soluciones por nosotros adoptadas.

1.1 Digitalización de Señales.

El nombre de señal analógica deriva de que al ser una señal analógica representa un hecho físico y tales circuitos que procesan dichas señales son conocidos como circuitos analógicos. Una forma alternativa de representar las señales es por medio de una secuencia numérica discretizada, cada número representa un valor de señal en cada instante de tiempo. La señal resultante es llamada como señal digital. Para ver como una señal puede ser representada en esa forma – como las señales analógicas se las puede convertir en señales digitales – En la figura 1.1 se representa una señal analógica en intervalos idénticos de tiempo marcados en la escala de tiempo [1].

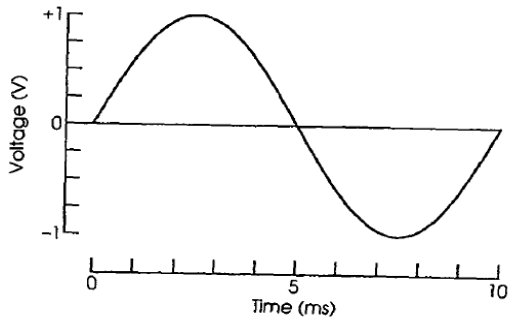

Figura 1.1: Señal Analógica

En cada uno de esos instantes, es medida la amplitud de la señal. Tal proceso es mostrado en la figura 1.2 que representa la señal de la figura 1.1 en los instantes de muestreo.

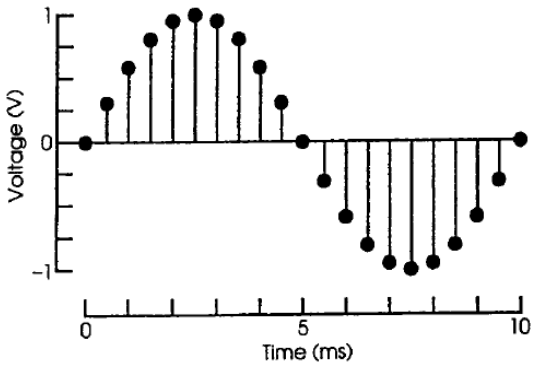

Figura 1.2: Muestras de una señal analógica.

Está ya no es más una función continua en el tiempo, ha pasado a ser una señal discreta en el tiempo.

Si representamos la amplitud de cada muestra de la señal en la figura 1.2 por un número de dígitos fijos, entonces la amplitud de la señal no será más continua; al contrario será considerada discretizada o digitalizada. Entonces la señal resultante será simplemente una secuencia de números que representan las amplitudes de las muestras sucesivas de la señal. A tal proceso se lo conoce como proceso de muestreo.

Escoger el sistema numérico adecuado para la representación de la señal por un número de dígitos finitos tiene un impacto significativo en la complejidad de los circuitos digitales necesarios al procesamiento de tales señales; esto permitirá potencializar la información obtenida y al mismo tiempo reemplazar circuitos digitales por analógicos.

Una vez que tales circuitos analógicos fueran reemplazados por los circuitos digitales, y que estos están presentes en cualquier equipo electrónico por sus ventajas de facilidad (orientación al software) y su confiabilidad (fidelidad de datos), aparecerán una serie aplicaciones que el trabajo con circuitos digitales permitirá realizar.

La primera ventaja se basa a la factibilidad de crear firmware o software embebido en el equipo que permite ser renovado, manteniendo el hardware existente, y efectuando cambios de funcionamiento. La segunda ventaja respecto a la fidelidad de sistemas analógicos, puede ser influenciada por tres factores: condiciones climáticas, ajustes de componentes, y envejecimiento. Ya que siempre es deseable que equipos procesen información de la misma manera en cualquier condición climática, aunque los componentes están influenciados por factores eléctricos que varían en función de la temperatura, humedad, presión y eso cambie el funcionamiento electrónico del sistema, mas no cambiará el procesamiento lógico del mismo, ya que por ejemplo si el sistema está en condiciones óptimas de funcionamiento electrónico arroja una señal lógica de 4,7v y bajo condiciones anormales del clima arroja 4,5v , a los dos valores se los encierra como nivel lógico. [12]

1.2 Digitalización de Señales de Pequeña Amplitud.

Las señales de pequeña amplitud tienen una dificultad adicional, para su digitalización y estudio, poseen una relativa dificultad a ser sorteada durante el desarrollo de este proyecto; la intromisión de señales indeseadas por efecto de redes eléctricas cercanas, equipos electrónicos presentes, y ondas electromagnéticas que viajan por el espacio; de tal forma que esas señales se mezclan con las señales por nosotros estudiadas malogrando nuestro objetivo. [7]

Usualmente las señales de baja o pequeña amplitud se encuentran en el rango de los mili-voltios (mV) y en muchos casos en el rango del micro-voltio (μV). La utilización de circuitos que nos permitan rechazar las señales indeseadas durante este proyecto será estudiado en el capítulo 3 donde se instruye respecto a un circuito de Electrocardiograma.

Para la construcción de los circuitos de adquisición de señales de pequeña amplitud será necesario la utilización de elementos propios para este fin, nos enfocaremos en amplificadores de instrumentación, y elementos que provean la precisión requerida para el canal de adquisición.

1.3 Constitución del Proyecto.

Este proyecto pretende crear un prototipo de monitor cardíaco el cual tenga la virtud de un canal de comunicación USB con el PC; el proyecto pretende implementar un canal analógico de adquisición, una interfase de adquisición mediante un conversor analógico digital y una interfase de comunicación USB, mediante un controlador de arquitectura ARM, para posteriormente mostrar los resultados en una aplicación para PC; como se muestra en la figura 1.3.

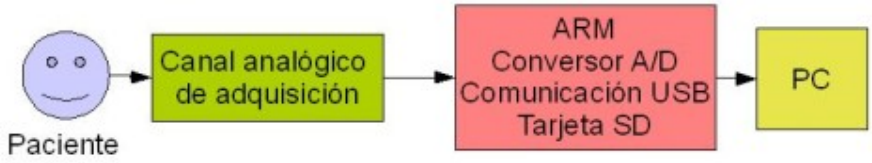

Figura 1.3 Diagrama de Bloques Prototipo ECG

El monitor ECG (*Electrocardiograma*) está constituido por las siguientes etapas: adquisición, digitalización, transmisión, almacenamiento y presentación de las señales obtenidas.

La etapa de adquisición es una etapa meramente analógica en la cual se han aplicado criterios de amplificación de señales de pequeña amplitud en la que la utilización y correcto dimensionamiento de los amplificadores de instrumentación será crucial para la amplificación total de todo el circuito analógico.

La etapa de filtraje priorizará la banda en la cual nuestra señal de electrocardiograma se encuentra ubicada, eliminando la mayor cantidad de ruido presente en el ambiente, intromisiones electromagnéticas, ruido blanco y de red eléctrica.

La segunda etapa es la de digitalización en la cual utilizando las virtudes de nuestro procesador ARM, utilizamos el conversor analógico digital que este posee, utilizando una

resolución de 10 bits que es configurable mediante *firmware.* Una vez que la información se encuentra digitalizada se transmitirá mediante registros internos del procesador hacia la etapa siguiente.

La próxima etapa de nuestro proyecto es la de transmisión que utiliza las virtudes del procesador ARM, y sus características de comunicación USB, para transmitir los datos obtenidos al computador y su posterior visualización, está comunicación USB se la realizó mediante la identificación de un dispositivo HID (*Human Interface Device*), dispositivo de interfase humana, el cual tiene la virtud de que Microsoft mediante su sistema operativo Windows posee las librerías para su caracterización lo cual lo hace un dispositivo manejable en el área de hardware y software.

La última etapa es la de visualización en la cual se presentan las señales obtenidas, en las anteriores etapas. Adicionalmente está etapa tiene la característica de identificar al dispositivo de comunicación y adicionalmente administrar la comunicación con el mismo. La elaboración de este software permite presentar los datos e informaciones relevantes para quien interactúe con el dispositivo de manera sencilla.

CAPÍTULO 2:

2. ATM91SAM7S256 E INTERFASE PC.

Se describirá la placa de desarrollo SAM7-P256, se iniciará conceptualizando la arquitectura de un procesador ARM y cuáles son sus características y una comparación con las arquitecturas Von Neumann y Harvard, adicionalmente se presentará cuáles son las características de comunicación con el PC, y se enfatizara en lo referente a la comunicación USB.

2.1 Descripción de la placa de dasarrollo SAM7-P256

Las placas de desarrollo son herramientas por medio de las cuáles se puede realizar en montaje de prototipos, ejecutar pruebas y realizar cambios durante el proceso de desarrollo. Contamos con placas que los propios fabricantes de los procesadores las elaboran, pero su costo llega a ser alto, y en consideración del nivel investigativo que tenemos a este nivel, no justifica realizar una inversión de ese tipo. Pero existen empresas que desarrollan placas de menor costo, con los mismos procesadores y de muy alta calidad; la principal diferencia entre ellas es que la ofrecida por los fabricantes utiliza elementos externos de bajo consumo y menor tiempo de respuesta, mientras que nuestra placa posee elementos comunes. Pero cumple con los requerimientos que se necesita. La placa por la cual se ha optado, es del fabricante americano OLIMEX, y se muestra en la figura 2.1.

Figura 2.1: Placa OLIMEX

2.1.1 Características de la placa de desarrollo.

La presente placa de desarrollo consta de los siguientes atributos, por los cuáles fue seleccionada.

- MCU AT91SAM7S256 16/32 bit con 256 bytes de memoria flash.
- Conector J-TAG 2x10, requerido para proceso de programación y depuración.
- Conector USB.
- Dos canales RS232.
- Conector de tarjetas SD/MMC.
- Dos Botones.
- Potenciómetro conectado al conversor ADC.
- Permisor conectado al conversor ADC.
- Dos leds de estatus.
- Regulador de voltaje a 3.3V y 800mA.
- Led de energía de la placa.
- Capacitor de filtraje de la fuente de poder.
- Circuito reset.
- Botón de reset.
- Cristal de 18.432MHz.
- Dimensiones 4.7x3.15” (80x120 mm). [13]

Como podemos apreciar, la tarjeta tiene importantes características que facilitarán la comunicación con el PC, durante la etapa de desarrollo. El conector que más interesó para el desarrollo del proyecto es el conector USB, ya que mediante este concurre la aplicación final del prototipo. Adicionalmente poseemos el conector J-TAG (*Joint Test Action Group*), que como ya lo hemos mencionado, es una importante herramienta de desarrollo, mediante la cual realizaremos la programación y depuración del *firmware*, conjuntamente con el cable J-TAG, que sus características las explicaremos más adelante. El conector de tarjetas MMC (Multimedia Card) , permitirá desarrollar un *firmware* que nos permita trabajar con este tipo de tarjetas que en el actual momento tiene mucha importancia en el mercado por su simplicidad y su capacidad.

2.3 Procesador ARM.

El diseño del ARM comenzó en 1983 como un proyecto de desarrollo en la empresa *Acorn Computers Ltd. Roger Wilson* y *Steve Furber* lideraban el equipo, cuya meta era, originalmente, el desarrollo de un procesador avanzado, pero con una arquitectura similar a la del MOS 6502. La razón era que *Acorn* tenía una larga línea de ordenadores personales basados en dicho micro, por lo que tenía sentido desarrollar uno con el que los desarrolladores se sintieran cómodos.

El equipo terminó el diseño preliminar y los primeros prototipos del procesador en el año 1985, al que llamaron ARM1. La primera versión utilizada comercialmente se bautizó como ARM2 y se lanzó en el año 1986.

El ARM2 es probablemente el procesador de 32 bits útil más simple del mundo, ya que posee sólo 30.000 transistores. Su simplicidad se debe a que no está basado en micro código (sistema que suele ocupar en torno a la cuarta parte de la cantidad total de transistores usados en un procesador) y a que, como era común en aquella época, no incluye caché. Gracias a esto, su consumo en energía es bastante bajo, a la vez que ofrece un mejor rendimiento que un procesador de la familia 80286. Su sucesor, el ARM3, incluye una pequeña memoria caché de 4 Kb, lo que mejora los accesos a memoria repetitivos. [1]

2.4 Arquitectura procesador ARM.

El micro controlador AT91SAM7S256, incorpora un procesador ARM7TDMI ARM Thumb el cual es basado en una arquitectura RISC de 32 bits (figura 2.2), con alta velocidad de procesamiento de hasta 55 MHz, y diversas prestaciones en periféricos, incluido un dispositivo USB 2.0 y otros que reducen la implementación de dispositivos externos.

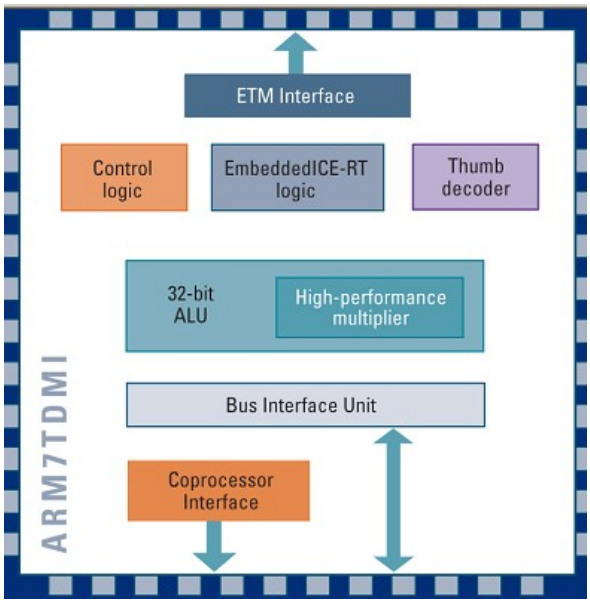

Figura 2.2: Arquitectura ARM

La arquitectura computacional, **RISC** de ***R**educed **I**nstruction **S**et **C**omputer* (Computadora con Conjunto de Instrucciones Reducido). Posee las siguientes características fundamentales:

1. Instrucciones de tamaño fijo , presentado en un reducido número de formatos.
2. Sólo las instrucciones de carga y almacenamiento acceden a la memoria por datos.

El objetivo de diseñar máquinas con está arquitectura es posibilitar la segmentación y el paralelismo en la ejecución de instrucciones y reducir los accesos a memoria. [4]

Las máquinas RISC protagonizan la tendencia actual de construcción de microprocesadores. PowerPC, DEC Alpha, MIPS, ARM, son ejemplos de algunos de ellos.

RISC es una filosofía de diseño de CPU para computadora que está a favor de conjuntos de instrucciones pequeños y simples que toman menor tiempo para ejecutarse. El tipo de procesador más comúnmente utilizado en equipos de escritorio, el Intel x86, está basado en CISC en lugar de RISC, aunque las versiones más nuevas traducen instrucciones basadas en CISC x86 a instrucciones más simples basadas en RISC para uso interno antes de su ejecución.

La idea fue inspirada por el hecho de que muchas de las características que eran incluidas en los diseños tradicionales de CPU para aumentar la velocidad estaban siendo ignoradas por los programas que eran ejecutados en ellas. Además, la velocidad del procesador en relación con la memoria de la computadora que accedía era cada vez más alta. Esto conllevó la aparición de numerosas técnicas para reducir el procesamiento dentro del CPU, así como de reducir el número total de accesos a memoria.

El procesador ARM también tiene algunas características que son particulares en otras arquitecturas también consideradas RISC, como el direccionamiento relativo, y el pre y post incremento en el modo de direccionamiento.

El la arquitectura del procesador ARM tiene dos modos de funcionamiento: el ARMI con instrucciones que ocupan 4 bytes, más rápidas y potentes (hay instrucciones que sólo están en este modo) pero con mayor consumo de memoria y de electricidad. Y el modo THUMB, más limitado, con instrucciones que ocupan 2 bytes y con menor consumo de corriente.

Quizás en parte por el uso condicional de usar 4 bits por cada instrucción, los procesadores más recientes traen una instrucción de 16 bits, llamada THUMB. Este pretende disminuir la cantidad de código escrito. Así como mejorar la densidad del código, el rendimiento puede ser superior a un código de 32 bits en donde el puerto de memoria o ancho del bus de comunicaciones son menores a 32 bits. [4]

El primer procesador con la tecnología **THUMB** fue el ARM7TDMI. Toda la familia posterior al ARM9, tienen incorporada la tecnología en su núcleo.

Los diseños RISC también prefieren utilizar como característica un modelo de memoria Harvard, donde los conjuntos de instrucciones y los conjuntos de datos están conceptualmente separados; esto significa que el modificar las direcciones donde el código se encuentra pudiera no tener efecto alguno en las instrucciones ejecutadas por el procesador (porque la CPU tiene separada la instrucción y el caché de datos, al menos mientras una instrucción especial de sincronización es utilizada). Por otra parte, esto permite que ambos caches sean accedidos separadamente, lo que puede en algunas ocasiones mejorar el rendimiento.

El dispositivo está provisto de una memoria tipo Flash-ROM (Flash *Read only Memorie* – Memoria de lectura rápida)embutida, la cual puede ser programada mediante una interfase paralela IEEE 1149.1 o mas conocida como J-TAG (*Joint Test Action Group*). Además tiene la característica de configuración de bit de seguridad, el cual protegerá al *firmware* de una sobre escritura accidental o para mantener la confidencialidad del mismo.

El procesador ATM91SAM7S256 tiene las siguientes características que son expuestas en la tabla 2.1, las que lo diferencian de sus similares de la misma familia.

Disp	**FLASH**	**Org. de Flash**	**SRAM**	**Disp USB**	**USA RT**	**Interup Exter.**	**Canales PDC.**	**Canales TC.**	**líneas I/O**	**Pakt**
sam7 s256	256k byte	Plano simple	64k byte	1	2	2	11	3	32	LQFP/Q FN 6

Tabla 2.1: Caracteristicas del procesador ATM91SAM7S256. [4]

Una importante característica de este dispositivo es su capacidad de comunicación con una interfase J-TAG, es un dispositivo de depuración de componentes electrónicos, lo que la convierte en una poderosa herramienta de desarrollo.

Este procesador posee una memoria flash de 256 kbytes, con un acceso rápido de 30 Mhz por ciclo. Posee la capacidad de 10,000 ciclos de escrituras y una capacidad de retención de 10 años. 16 bits de seguridad, para 16 sectores de memoria en 64 páginas. Adicionalmente posee

una memoria de acceso rápido SRAM de 64kbites.[4]

La operación interna del dispositivo se la realiza mediante el manejo de módulos los cuáles requerirán señales de reloj y energía que debe ser habilitado mediante *firmware* según requerimientos específicos. Por tanto el dispositivo posee los siguientes elementos

1. Generador de señales de reloj.
2. Controlador de energía.
3. Controlador Avanzado de interrupciones (AIC).
4. Unidad de *Debug*.
5. Controlador de interfase de periféricos seriales (SPI).
6. Controlador de interfase Two-wire (TWI).
7. Controlador USART.
8. Controlador Serial Sincrono (SSC).
9. Timer Counter (TC).
10. Controlador PWM.
11. Dispositivo USB (USB Device Port, UDP).
12. Conversor analógico digital (ADC).

2.5 Arquitectura Von Neuman y Harvare

La arquitectura Von Neumann se refiere a las arquitecturas de computadoras que utilizan el mismo dispositivo de almacenamiento tanto para las instrucciones como para los datos (a diferencia de la arquitectura Harvard). El término se acuñó en el documento *First Draft of a Report on the EDVAC* (*Electronic Discrete varíable Automatic Computer*), escrito por el conocido matemático John Von Neumann, que propuso el concepto de programa almacenado. Dicho documento fue redactado en vistas a la construcción del sucesor de la computadora ENIAC.[1]

Los ordenadores con arquitectura Von Neumann constan de cinco partes: La unidad aritmético-lógica o ALU, la unidad de control, la memoria, un dispositivo de entrada/salida y el bus de datos que proporciona un medio de transporte de los datos entre las distintas partes.

Un ordenador con arquitectura Von Neumann realiza o emula los siguientes pasos secuenciales:

1. Enciende el ordenador y obtiene la siguiente instrucción desde la memoria en la dirección indicada por el contador de programa y la guarda en el registro de instrucción.
2. Aumenta el contador de programa en la longitud de la instrucción para apuntar a la siguiente.
3. Decodifica la instrucción mediante la unidad de control. Ésta se encarga de coordinar el resto de componentes del ordenador para realizar una función determinada.
4. Se ejecuta la instrucción. Ésta puede cambiar el valor del contador del programa, permitiendo así operaciones repetitivas. El contador puede cambiar también cuando se cumpla una cierta condición aritmética, haciendo que el ordenador pueda 'tomar decisiones', que pueden alcanzar cualquier grado de complejidad, mediante la aritmética y lógica anteriores.
5. Punto 2.

2.6 Arquitectura Harvard

El término Arquitectura Harvard originalmente se refería a las arquitecturas de computadoras que utilizaban dispositivos de almacenamiento físicamente separados para las instrucciones y para los datos (en oposición a la Arquitectura Von Neumann). El término proviene de la computadora Harvard Mark I, que almacenaba las instrucciones en cintas perforadas y los datos en interruptores.

Todas las computadoras constan principalmente de dos partes, la CPU que procesa los datos, y la memoria que guarda los datos. Cuando hablamos de memoria manejamos dos parámetros, los datos en sí, y el lugar donde se encuentran almacenados (o dirección).

En los últimos años la velocidad de las CPUs ha aumentado mucho en comparación a la de las memorias con las que trabaja, así que se debe poner mucha atención en reducir el número de veces que se accede a ella para mantener el rendimiento. Si, por ejemplo, cada instrucción ejecutada en la CPU requiere un acceso a la memoria, no se gana nada incrementando la velocidad de la CPU - este problema es conocido como 'limitación de memoria'.

Se puede fabricar memoria mucho más rápida, pero a costa de un precio muy alto. La solución, por tanto, es proporcionar una pequeña cantidad de memoria muy rápida conocida con el nombre de caché. Mientras los datos que necesita el procesador estén en la caché, el rendimiento será mucho mayor que si la caché tiene que obtener primero los datos de la memoria principal. La optimización de la caché es un tema muy importante de cara al diseño de computadoras.

La arquitectura Harvard ofrece una solución particular a este problema. Las instrucciones y los datos se almacenan en cachés separadas para mejorar el rendimiento. Por otro lado, tiene el inconveniente de tener que dividir la cantidad de caché entre los dos, por lo que funciona mejor sólo cuando la frecuencia de lectura de instrucciones y de datos es aproximadamente la misma. Está arquitectura suele utilizarse en circuitos impresos dedicados para procesamiento digital se señales, usados habitualmente en productos para procesamiento de audio y video.

Con está breve introducción respecto a la arquitectura de procesadores, se ha logrado comprender que el procesador at91sam7, es un dispositivo de alto desempeño gracias a las virtudes de poseer un procesador ARM, el cual se basa en una arquitectura RISC de 32-bits, con características del modelo Hardvard. Este procesador ya pertenece a la generación TDMI (*Techniciens Demolition Maconnerie Iseroise*) por lo que posee un grupo de instrucciones del tipo THUMB, lo que lo hace aun más rápido en sus procesos. [1]

2.6 Comunicación USB.

Características Generales.

La comunicación USB (*Universal Serial Bus*), desde que fue lanzada, se ha convertido en un patrón en la microelectrónica. Su desarrollo partió de la colaboración de diversas empresas líderes del mercado en las áreas de *software* y *hardware*. Su popularidad se debe a diversas virtudes.

- La facilidad de uso: en la gran mayoría de los casos no es necesario la configuración de *hardware* para su funcionamiento, solo basta conectarlo y el *host* mismo se encarga de reconocer la conexión eléctrica del dispositivo.
- Velocidad: altas tasas de comunicación sin formarse un cuello de botella.
- Confiabilidad: los errores que reproducen son extraños y cuando ocurren son

rápidamente corregidos por los algoritmos implantados con el padrón.

- Flexibilidad: una grande variedad de periféricos y dispositivos pueden ser utilizadas en la comunicación USB.
- Precio: costos bajos de componentes han dado que los fabricantes implementen está interfase en sus equipos.
- Soporte de los sistemas operacionales: los desarrolladores en la mayoría de las veces no necesitan escribir *drivers* de bajo nivel para que los dispositivos utilicen este tipo de comunicación. [1]

El patrón para los usuarios finales se muestra bastante sencillo. Esta interfase permite utilizar hasta 127 dispositivos diferentes que podrán ser conectados a través de *hubs*. El proceso automático de configuración permite que al conectar el dispositivo o periférico, el *host* comunica al sistema operacional y este determina la configuración, por lo que no es necesario reiniciar el sistema antes de usar el dispositivo.

La comunicación USB soporta tres velocidades de acuerdo con la versión. La versión 1.0 (*low-speed*) maneja una tasa de 1,5Mbps. La versión 1.1 (*full-speed*) maneja una tasa de 12Mbps. Y la versión 2.0 (*high-speed*) maneja una tasa de 480Mbps. Todo micro computador compatible con USB opera por patrón en las versiones 1.0 e 1.1. La versión 2.0 requiere un *hardware* específico presente en la placa madre del micro controlador o en la placa de expansión. [1]

El soporte ofrecido por los sistemas operacionales consiste en detectar al dispositivo al ser conectado o desconectado del sistema, comunicar con los dispositivos recién detectados para determinar la velocidad, el modo de operación, la cantidad de datos transmitidos y proveer un mecanismo que permita que los programas de *drives* se comuniquen con el controlador *host.*

La empresa *Microsoft* disponibilizo para su sistema operacional una serie de paquetes de *drivers* para diversos dispositivos, como dispositivos de audio, módem, cámaras, *scanner,* impresoras, dispositivos de almacenamiento y dispositivos de interacción humana como teclados, *Mouse*, y controles. Ese último es conocido como HID (*Human Interface Device*) y facilita mucho el desarrollo de dispositivos debido a su simplicidad comparada con las demás clases.

Cada dispositivo que utiliza comunicación USB debe poseer un micro controlador que

manipule el protocolo de comunicación que tiene un grado relativamente elevado de complejidad. Algunos fabricantes de circuitos integrados desarrollaron dispositivos conocidos como puentes que implementan el protocolo de comunicación, mediante la generación de señales que facilitan el desarrollo de *hardware.*

La comunicación USB fue proyectada para operar con comunicación serial asíncrono, osea sin la utilización de una señal de reloj de referencia y con señales de datos diferenciales. Para mantener el sincronismo de las dos señales entre el transmisor y el receptor es utilizada la codificación NRZI (*Non-Return to Zero Inverted*). La distancia máxima del cable de comunicación debe de ser un máximo de cinco metros. Este debe tener la característica de ser blindado y compuesto por los conductores en la tabla 2.2.

Pin	Señal	Color
1	VBus (+5V)	Rojo
2	D-	Blanco
3	D+	Verde
4	GND	Negro

Tabla 2.2: Conductores cable USB.[1]

Cada dispositivo USB, por la especificación, debe implementar subdispositivos que respondan eléctricamente a las transacciones. Tales subdispositivos son llamados *Endpoints*. Cada *endpoint* recibe una dirección única que es compuesta por un número y por la dirección de destino de los datos, que se envía o se recibe. Todo dispositivo debe tener un *Endpoint* de número *cero* que realiza las transacciones de control y que por tanto envía y recibe datos, en realidad este *endpoint* está formado por dos *endpoints,* uno de recepción y otro de transmisión, en la misma dirección. Por lo que se ha dispuesto para el direccionamiento de cada *endpoint* cuatro bits para dirección*,* 3 bits reservados y un bit de identificación, con un total de *8* bits*;* y para atributos dos bits de control, y 6 bits reservados, así con un total de dieciséis bits incluyendo el número cero, para cada *endpoint*.

La transferencia da datos en el patrón USB es realizada a través de paquetes de datos. Esos paquetes son del tipo *token*, *data* y *handshake.*

Los principales paquetes del tipo *token* son paquetes *setup*, que son utilizados para indicar que

el próximo paquete *data* será para escribir en el *endpoint cero*; paquetes de entrada tipo *in* que son utilizados para que el próximo paquete de datos sea leído por el *endpoint* de salida y; paquetes de salida del tipo *out* que son utilizados para que el próximo paquete de datos sea escrito en el *endpoint* de entrada.

Los paquetes tipo *data*, pueden ser del tipo *data1* o *data0,* e son efectivamente los datos transportados del *host* para el dispositivo, o del dispositivo para el *host*, de acuerdo con el paquete *token* que presidió.

Los paquetes del tipo *handshake* pueden ser del tipo *ack, nak* y *stall.* El *ack* es utilizado para informar al receptor, *host* o dispositivo, que el dato fue recibido correctamente. O *ack* es utilizado para informar que el *host* o dispositivo, recibió un paquete de datos correctamente. O *nack* es utilizado para informar que el *host* o dispositivo, está ocupado o inoperante. El *stall* es utilizado para informar un error en la comunicación.

La comunicación entre un dispositivo USB y el *host* es realizada mediante transacciones. Una transacción es un conjunto de paquetes enviados y recibidos sin errores. Pueden ser de cuatro tipos: *control, bulk, interrupt* e *isochronous*. Las transacciones del tipo *control,* deben ser implementadas para todos los dispositivos USB y tienen la finalidad de permitir al dispositivo *host* el envió de solicitudes de información sobre el dispositivo y leer esas informaciones. Las transacciones del tipo *bulk,* pueden ser implementadas apenas en los dispositivos que operan en las versiones 1.1 (*full-speed*) y 2.0 (*high-speed*) y son utilizados cuando la tasa de transferencia de datos no es crítica, esto es, cuando el dispositivo o el *host* pueden aguardar que termine de realizar la transacción que se está ejecutando. Las transacciones del tipo *interrupt* pueden ser implementadas por cualquier dispositivo USB, y son utilizadas cuando el dispositivo o el *host* necesitan de atención periódica como un *Mouse,* o un teclado. Las transacciones del tipo *isochronous* pueden ser implementadas apenas en dispositivos que operan en las versiones 1.1 y 2.0 son utilizadas en dispositivos que necesitan de la garantía del tiempo de datos, pero pueden tolerar errores ocasionales.[1]

2.7 Proceso de Enumeración.

La enumeración es el intercambio inicial de informaciones entre el dispositivo y el *host,* cuya finalidad es la de identificar el dispositivo y atribuir a este dispositivo un *driver* y su posibilidad de comunicación con aplicaciones.

El proceso de enumeración es realizado a través de transacciones del tipo *control* mediante el *endpoint 0* e incluye la atribución de direcciones al dispositivo, la lectura de estructuras de datos que caracterizan al dispositivo, la atribución, la carga de un *device driver* y la selección de configuraciones de acuerdo con los datos recibidos, que incluyen el tipo de transacciones y los *endpoints* que son utilizados. [2]

Uno de las etapas del proceso de enumeración es la detección de la conexión o remoción del dispositivo. Cuando el sistema es inicializado, se ejecuta una inspección de los dispositivos USB que se encontrasen, y después con una cierta periodicidad realizara inspecciones para encontrar cambios en los dispositivos.

Cuando un dispositivo es detectado el *hub* comunica al *host* y este envía una serie de comandos al dispositivo, para que este dispositivo sea enumerado debe responder a cada comando enviado por el *host.*

La idea del proceso de enumeración es que sea transparente al usuario final y ocurra de manera automática. En el ambiente Windows apenas un mensaje es mostrado y dependiendo del tipo de dispositivo es necesario cargar un *device driver* en la primera ocasión que ocurre la conexión.

Los pasos del proceso de enumeración son los siguientes:

1. El dispositivo es conectado a un puerto USB: el *hub* le provee alimentación al dispositivo y este pasa a estar en estado alimentado.
2. El hub detecta el dispositivo: las tensiones en las líneas de señal son monitoreadas a partir de dos resistores de *pull-down* en el *hub*. Cuando hay un cambio ocasionado por un resistor de *pull-up* en el lado del dispositivo de la línea D+ es reconocido un dispositivo versión 1.1 (*full-speed*) o 2.0 (*high-speed*). Cuando el resistor *pull-up* en el lado del dispositivo está en la línea D- es reconocido un dispositivo versión 1.0 (*low-speed*). En esta fase entretanto no es aun detectada la versión del dispositivo, apenas el cambio de estado de las líneas de señal. El esquema de conexión se muestra en la figura 2.3.
3. El *host* reconoce el nuevo dispositivo: el *hub* raíz relata un evento al *host* e repasa en cualquier puerta o *hub*.

4. El *hub* detecta la versión USB soportada por el dispositivo: antes de que el dispositivo sea inicializado el *hub* detecta la versión USB y por tanto la velocidad del dispositivo e informa al *host*.
5. El *hub* inicializa el dispositivo: el *host* envía un comando al *hub* para inicializar el puerto donde fue detectado la presencia del nuevo dispositivo. El *hub* coloca las líneas de datos en la condición de *reset* que apenas es percibida por el dispositivo ya que su duración de 10ms.
6. El *host* verifica si la versión 1.1 del dispositivo soporta la versión 2.0; algunas señales especiales son enviadas al dispositivo y si este consigue interpretar y responder a esas señales todas las próximas transacciones son asumidas en la versión 2.0.
7. El *hub* establece un camino entre el dispositivo y la comunicación: el *host* consulta al *hub* si el dispositivo ya salió del *reset*. Esa consulta es realizada hasta que la respuesta sea positiva. Cuando el dispositivo sale de la condición de *reset*, está listo para comunicarse a partir del *endpoint 0* y transacciones de control *host.* En esa etapa el dispositivo puede consumir hasta 100mA.
8. El *host* consulta el tamaño máximo del paquete de datos utilizado por el dispositivo: el *host* envía un requerimiento al dispositivo e aguarda un descriptor el cual contiene el tamaño máximo del paquete utilizado por el *endpoint 0* que está presente en el octavo bit del descriptor.
9. El *host* atribuye una dirección al dispositivo: el *host* envía un comando al dispositivo indicando a este cual es su dirección durante este conectado al sistema. El dispositivo envía una confirmación al *host* y almacena internamente su dirección atribuida.
10. El *host* identifica las funcionalidades del dispositivo: el *host* solicita a través de la dirección atribuida, informaciones las funcionalidades son disponibilizadas y el dispositivo responde enviando un descriptor de dispositivo que simplemente es una estructura de datos estandarizados. El *host* solicita los descriptores de configuración que indican información respecto a los *endpoints* que serán utilizados, de que tipo son, y el tamaño de los paquetes, etc. El dispositivo responde entregándole los descriptores de configuración solicitados.
11. El *host* atribuye y carga un *device driver:* el *host* con base a los datos del fabricante y producto recibidos por el descriptor de dispositivos indica al sistema cual *device driver* fue atribuido al dispositivo y solicita que sea cargado en memoria.
12. El *device driver* selecciona las configuraciones: el *host* envía comandos de configuración al dispositivo de acuerdo con las solicitaciones de *device driver.* En ese punto el dispositivo está enumerado y listo par usar.

La remoción del dispositivo es detectada por el *hub* que desconecta el puerto del dispositivo e indica al *host* lo ocurrido. Este indica al sistema que el dispositivo no está más disponible, y solicita la descarga del *device driver* y libera la dirección utilizada. [2]

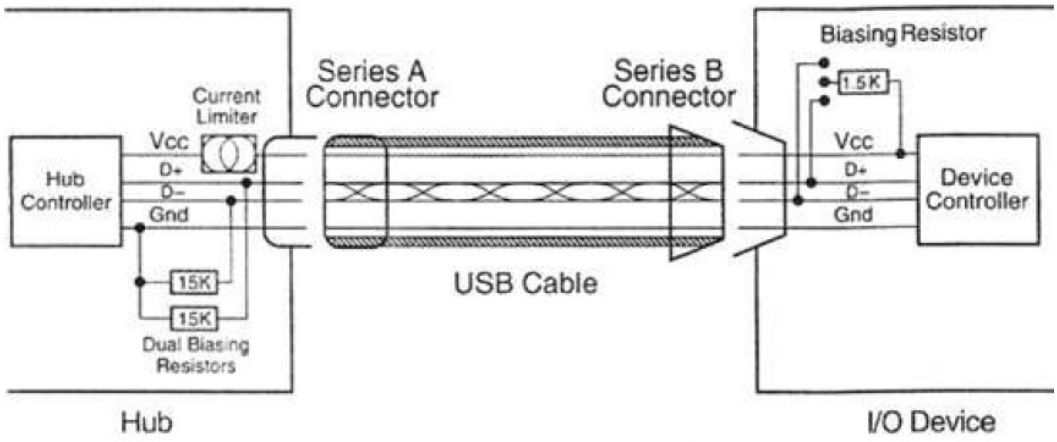

Figura 2.3: Esquema de conexión USB [2]

2.7.1 Tipos de Descriptores

Como se vio en el proceso de enumeración, los descriptores son estructuras de datos, o sea, bloques de información formados, que son enviados del dispositivo al *host* como respuestas a una petición.

A la medida que el proceso de enumeración ocurre, los descriptores son solicitados, primero son pedidos los descriptores que detallan al dispositivo de manera general, después van pasando los descriptores que detallan la configuración y después los de cada interfase de configuración y cada interfase de *endpoint*.

Cada dispositivo debe poseer apenas un descriptor de dispositivos que contiene información sobre el mismo y especifica el número de configuraciones soportadas por este. Cada dispositivo tiene uno o más descriptores de configuración, en las cuáles podría constar información de consumo de potencia y el número de interfaces de configuración.

Los descriptores pueden ser de los siguientes tipos:

- **Descriptor de dispositivo:** ese descriptor es obligatorio para cualquier dispositivo. El contiene información básica sobre dispositivos, y es el primero a

ser solicitado y permite que el *host* posea toda la información adicional al dispositivo. Este descriptor informa la versión USB soportada, cual clase, subclase, producto y fabricante, cual es el tamaño máximo de los paquetes utilizados en el *Endpoint 0*, cuáles son las configuraciones posibles e informa también cuáles son los descriptores *string* relacionados al fabricante, descripción y número de serie.

- **Descriptor cualificador de dispositivo:** ese descriptor es obligatorio para todos los dispositivos versiones 1.1 o 2.0. Ese descriptor almacena la información de la velocidad que actualmente no está en uso y es alterado cuando hay un cambio en la versión USB.
- **Descriptor de configuración:** ese descriptor es obligatorio para cualquier dispositivo de configuración contiene informaciones sobre las funcionalidades del dispositivo, como consumo de potencia e número de interfaces soportadas. Apenas un descriptor de configuración puede estar siendo utilizado al momento.
- **Descriptor de configuración de otra velocidad:** ese descriptor es obligatorio para todos los dispositivos que soportan ambas versiones 2.0 y 1.1. El descriptor contiene el mismo tipo de información que el descriptor de configuración para la velocidad que no está siendo utilizado.
- **Descriptor de interfase:** ese descriptor es obligatorio para cualquier dispositivo. El descriptor contiene información sobre la cantidad de *endpoints* de la interfase, el identificador de la interfase y de la clase, subclase y protocolo de interfase. Un dispositivo puede tener varías interfases activas al mismo momento o mutuamente exclusivas.
- **Descriptor de *Endpoint:*** ese descriptor es opcional para cualquier dispositivo. Ese descriptor posee una dirección de *endpoint,* la dirección de flujo de datos, el tipo de transacción utilizada, el tamaño máximo de cada paquete y el intervalo de tiempo de vigencia. El *endpoint 0* no posee descriptores porque es obligatorio, se asume que trabaja con transacciones de control y la información necesaria para el mismo es transmitidas por el descriptor de dispositivo.
- **Descriptor de *String:*** ese descriptor es opcional para cualquier dispositivo. Contiene información textual utilizada para completar la información de cada uno de los demás descriptores. [3]

Existen diversos descriptores que son utilizados de acuerdo con las especificaciones del dispositivo, aquí fueron presentados apenas los descriptores patrón.

***PC Request* Patrones.**

PC *Request* son todos los pedidos enviados por el *host* mediante un canal de control, normalmente por el *endpoint 0* que es obligatorio en todo dispositivo. Este canal de control está siempre abierto y operacional. El dispositivo deberá responder a todos los pedidos enumerados en la tabla 2.3.

Los pedidos son totalmente definidos por dos parámetros, el *bmRequestType y el bRequest.* El parámetro *bmRequestType* es mapeado bit a bit de la siguiente manera: el bit 7 define la dirección de la fase de datos que sigue a la fase de *setup,* los bits 6 y 5 definen el grupo al cual el pedido pertenece e los bits 4 hasta el 0 definen el destinatario del pedido. Los posibles valores pueden ser vistos en la figura 2.4, los valores no mostrados son reservados. [1]

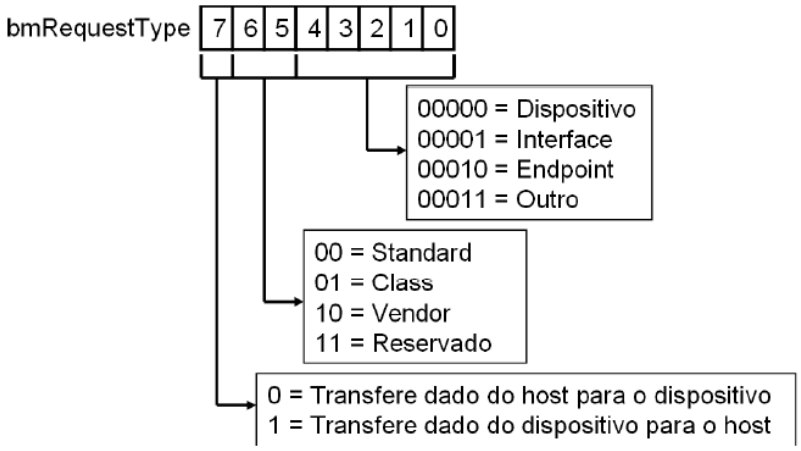

Figura 2.4: Esquema de bmRequestType

bRequest	Request
0	GET_STATUS
1	CLEAR_FEATURE
2	Reservado
3	SET_FEATURE
4	Reservado
5	SET_ADDRESS
6	GET_DESCRIPTOR
7	SET_DESCRIPTOR
8	GET_CONFIGURATION
9	SET_CONFIGURATION
10	GET_INTERFACE
11	SET_INTERFACE
12	SYNCH_FRAME

Tabla 2.3: Tabla de bmRequestType [3]

Ya el parámetro *bRequestes* un valor correspondiente al pedido, como se determina en la tabla anterior.

El proceso de enumeración se ejecuta mediante estos pedidos de la siguiente forma:

1. Request: Get_Port_Status. Para : *hub.*

   El *host* detecta un nuevo dispositivo.

2. Request: Set_Port_Feature(C_PORT_CONNECTION). Para: *hub*.

   Limpia la bandera indicando un cambio en el puerto del *hub*

3. Request: Set_Port_Feature (PORT_RESET.) Para: *hub*.

   El *hub* responde mandando un reset para el dispositivo. El mantiene el *reset* por 10 ms habilita el puerto. El PC *host* va detectar esa mudanza en el próximo periodo *polling*. Es durante ese tiempo de *reset* que el *hub* configura la velocidad del dispositivo. Si detecta una tensión alta en D-, configura el dispositivo *low-speed*. Si detecta una tensión alta en D+, configura el dispositivo *full-speed*. Si el dispositivo

soporta *high-speed*, es configurado como *full-speed* y dentro de los 10ms de *reset* manda una secuencia de datos en alta velocidad. Se el *hub* soporta *high-speed*, va a detectar esa secuencia y responder al dispositivo, configurándolo como *high-speed*. Caso contrario, el *hub* no consigue detectar esa secuencia de datos y no responderá, haciendo que el dispositivo permanezca configurado como *full-speed.*

4. Request: Get_Port_Status. Para: *hub*

   El *hub* espera hasta que el dispositivo haya retornado del estado de *reset.*

5. Request: Clear_Port_Feature(C_PORT_RESET). Para: *hub*

   Limpia las banderas en el registrador del *hub*. En ese punto el dispositivo está alimentado y ya fue reiniciado, después se encontrará en el estado de *default.* El dispositivo a partir de este momento va responder a los pedidos enviados a la dirección 0 (dirección patrón). Como el proceso de enumeración ocurre de forma exclusiva apenas un dispositivo responderá por la dirección patrón.

6. Request: Get_Device_Descriptor: Para: dispositivo.

   El dispositivo responde enviando al descriptor de dispositivo.

7. Request: Set_Address. Para: dispositivo.

   El PC *host* asigna una dirección al dispositivo. A partir de ese momento todos los pedidos para ese dispositivo serán enviados a esa dirección. El dispositivo debe guardar la dirección y responder a los pedidos que se le realicen.

8. Request: Get_Device_Descriptor. Para: dispositivo.

   El PC *host* repite ese pedido para la nueva dirección como verificación. Debe obtener la misma respuesta obtenida en el pedido 6.

9. Request: Get_Configuration_Descriptor. Para: dispositivo.

   El PC *host* comienza a recolectar mas información sobre el dispositivo, sus

configuraciones, interfases y *endpoints*. En ese punto puede ser necesaria la intervención del usuario.

10. Selección de *driver.*

    El PC *host* inicia la búsqueda de *drivers* para el dispositivo. Primeramente, intenta encontrar un archivo .INF con *VendorID* y *ProductID* equivalentes a los recolectados durante la enumeración. Si no fuera encontrado, va analizar si el dispositivo se encaja en una clase especifica, y caso consiga va cargar los *drivers* del USB patrón. Caso no tenga suceso en ninguna de estas tentativas el usuario será consultado por los *drivers* del dispositivo.

11. Request: Set_Configuration. Para: dispositivo.

    El dispositivo se encuentra configurado y está en estado operacional, pasando por el estado *configured*. [3]

2.8 Clases.

Siguiendo la filosofía USB de universalizar el tornar mas simple la inicialización de dispositivos en el computador. La comunicación USB debe ofrecer los más diversos tipos de dispositivos. Muchos de estos poseen características semejantes, pudiendo ser divididos en grupos: impresoras, *pens-drives*, *mouses*, maquinas fotográficas, etc. De esta manera toma sentido definir especificaciones para estos grupos de manera que ellos se comuniquen uniformemente con el *host*. Estos "grupos" son las clases, que ofrecen diversas ventajas tanto al usuario cuanto a los desarrolladores.

Los sistemas operacionales modernos ya tienen incluso *drivers* para la mayor parte de las clases definidas por USB-IF. Esto libera a los fabricantes de crear *drivers* específicos para sus productos. Los usuarios se liberan de utilizar CDs que terminan obsoletos, ya que basta conectar al puerto USB disponible que la instalación se da de forma automática y transparente para el usuario.

Cuando el dispositivo se encuadra en una clase pero presenta características que no son soportadas por ninguna clase patrón, el fabricante tiene la opción de desarrollar apenas un *driver* complemento, que ofrezca soporte para esas características.

Así como la especificación USB, cada clase define sus descriptores y pedidos específicos, bien como formato de datos intercambiados con el *host* y el dispositivo. A fin de posibilitar *drivers* más apropiados las especificaciones de cada tipo de equipo, a las clases se las ha subdividido en subclases con sus propios descriptores y pedidos. Estas subclases son parte de la especificación de las clases. [3]

Una lista completa de las clases y sus características puede ser obtenida en Http://www.usb.org/developers/devclass_docs.

En este documento nos remitiremos a la clase HID (*Human Interface Device*), ya que nuestro prototipo fue adaptado a las características de esta clase.

HID (*Human Interface Device*).

Las especificaciones de la clase HID ha sido claramente realizada para dispositivos de interacción tales como teclados, *mouses;* la verdad es que esta clase no sirve exclusivamente para estos propósitos. Un dispositivo HID debe apenas consumir o enviar cantidades pequeñas de datos a tasas inconstantes. Como la respuesta humana es demasiado lenta comparada a la velocidad del computador moderno, esta definición es suficiente para la mayor parte de los equipos de interacción humana. Buenos ejemplos del dispositivo HID sin interfase humana son termómetros digitales, lectores de código de barra y multimetros. [1]

Los dispositivos HID tienen tasas de transferencia de hasta 800 bps en *low-speed,* 64 Kbps en *full-speed,* 24,576Mbps en *high-speed*. Estas transferencias se dan por medio de los llamados *Reports*. Estos son bloques de datos sin restricciones de tamaño, con formato definido por el *Report descriptor,* un descriptor especifico de la clase HID. [5]

Un dispositivo HID posee las siguientes características:

Los datos serán transferidos mediante una estructura llamada *report.* El formato de un *report* es totalmente flexible y se puede ajustar a practicar a cualquier tipo de dato.

El *host envía* y recibe los datos (*reports*) utilizando apenas transacciones de tipo *control* o *interrupt.*

La cantidad máxima de datos en cada transacción varía de acuerdo con la versión USB, como ya se lo mencionó. Caso exista la necesidad de transferir un cantidad grande de datos se puede utilizar múltiples transacciones.

No hay garantía de tasa de transferencia fija. Por ejemplo, para un dispositivo configurado para un intervalo de *polling* de 10ms, el tiempo entre las transacciones será cualquier periodo igual o menor a 10ms, por lo que no existe una tasa transmisión fija. [5]

Como el *host* identifica un dispositivo HID.

El *host* identifica un dispositivo de la clase HID, durante el proceso de enumeración, a través de los descriptores. El descriptor de interfase contiene el campo Clase, donde deberá ser indicado el código de la clase. En el caso de la clase HID debe ser utilizado el código 03. Sabiendo esto, el *host* espera recibir también el descriptor HID, que contiene información al respecto de la versión HID, y sobre el descriptor de *report.* En la tabla 2.4 se muestra con detalles las características del descriptor HID. [1]

Offset (decimal)	Campo	Tamaño(bytes)
0	Inicio o saludo	1
1	Tipo de descriptor	1
2	Versión HID	2
4	Código del país	1
5	número de descriptores	1
6	Tipo de descriptores	1
7	Tipo de el descriptor anterior	2
9	Tipo descriptor (opcional)	1
10	Tam. descriptor (opcional)	2

Tabla 2.4 Descriptores clase HID

El campo “saludo” informa el número total de bytes del descriptor HID, tiene como mínimo 9 bytes.

El campo “Tipo de descriptor” debe tener el valor de 21H, e indica que se trata de un descriptor HID.

El campo “Versión HID” trata de la versión de la especificación HID. Para la versión 1.0 el

valor debe ser 0100H, mientras para la versión 0110H.

El campo “número de descriptores” informa el número de descriptores de la clase subordinados.

El campo “tipo de descriptor de clase” identifica el dispositivo posee un descriptor del tipo *report* o *physical*. Todo dispositivo HID debe poseer al menos un descriptor *report*.

El campo “Tamaño del descriptor anterior” son opciones y son usadas en caso existan otros descriptores de clase subordinados. Indicaran respectivamente el tipo y tamaño (en bytes) de cada descriptor.[1]

Descriptor *Report*

Un dispositivo HID se comunica con el *host* utilizando bloques de datos llamados *reports*. El descriptor de *report* contiene información respecto del formato de datos que el dispositivo enviara o recibirá. Par tener una idea mas clara respecto al descriptor de *report*, la tabla 2.5 muestra el descriptor creado para este proyecto, con los comentarios sobre cada campo. [1]

Seudo código	Ítem
06H,00H,FFH	USAGE_PAGE(Vendor Defined Page 1)
09H,01H	USAGE (Vendor usage 1)
A1H, 01H	COLLECTION (Aplication)
19H,01H	Usage Minimun (User defined)
29H, 08H	Usage Maximun (User Defined)
15H, 80H	LOGICAL_MAXIMUN (-128)
25H, 7FH	LOGICAL_MAXIMUN (127)
75H, 08H	REPORT_SIZE[8]
95H, 40H	REPORT_COUNT(64)
81H, 02H	INPUT (Data, Var, Abs)
19H, 01H	Usage Minimun (User Defined)
29H, 08H	Usage Maximun (User Defined)
91H, 02H	OUTPUT (Data, Var, Abs)
COH	END_COLLECTION

Tabla 2.5: Tabla Report

Cada ítem en el descriptor *report* consiste en un *byte* que lo identifica.

El ítem *Usage Page* es identificado por el valor 06H y especifica la función del dispositivo. La especificación HID lista los varios valores posibles para este ítem Algunos ejemplos serían de

control genérico para *desktop,* control para juegos, *displays* alfanuméricos y muchos otros. En el caso de nuestro prototipo el valor FF00H indica que la función del dispositivo es definida por el fabricante.

El ítem es identificado por el valor 09H y puede ser visto como un sub item del *Usage Page*. Suponiendo que el *Usage Page* tuviera el valor par un control genérico de *desktop*, su sub item (*Usage*) podría asumir valores para un Mouse, teclado o *joystick*. En el caso de nuestro prototipo el valor 01H indica el ser definido por el fabricante.

El ítem *Cellection* es identificado por el valor A1H y es a partir de ese punto que comienza a ser definido el formato del *report.*

Los ítems *Logical Minimmun* y *Logical Maximun* son identificados respectivamente por los valores 15H y 25H. Especifican franjas de valores que los datos pueden asumir, en complemento de dos. En este ejemplo, los datos pueden asumir valores de 80H (-128) hasta 7FH (+127), en valores decimales los valores van de 0 (00H) hasta 255 (FFH).

El ítem *Report Size* es identificado por el valor 75H e informa cuantos bits existen en cada dato. En este caso tendrá ocho (08H) bits, o sea un *byte.*

El ítem *Report Size* es identificado por el valor 95H e informa cuantos datos cada *report* tendrá. En este ejemplo, cada *report* tendrá 64 datos, cada uno de 8 *bits*.

Los ítems *Input* y *Output* son identificados respectivamente por los valores 81H y 91H e informa que los *reports* son bi direccionales (enviados del dispositivo para el *host* y viceversa)

El descriptor de *report* implementado para este proyecto es mostrar en este ejemplo es relativamente simple y corto. Además, los descriptores de este tipo pueden contener varios *ítems* adicionales, con unas variedades inmensas de datos y *report* diferentes. [1]

2.9 Tarjetas MMC y SD.

Las tarjetas de memoria SD (*Secure Digital Memory Card*) son las tarjetas de memoria más populares para equipos móviles. Las tarjetas SD poseen una superior compatibilidad que las tarjetas MMC (Multi Media Card). El manejo de la tarjeta de memoria se lo realiza bajo los comandos de lectura, escritura, borrado y error de control; los que serán completados dentro de la tarjeta de memoria. Los datos son transferidos entre la tarjeta de memoria y el controlador por unidades o por bloques de 512 bytes, esto puede ser visto como el manejo de un disco duro genérico. Que es su mayoría estará definido como un sistema de archivos tipo FAT12/16. Donde la definición de FAT32 solo se da para capacidades superiores a 4Gbytes.

La superficie de contacto con la tarjeta SD y MMC consta en la figura 2.5.

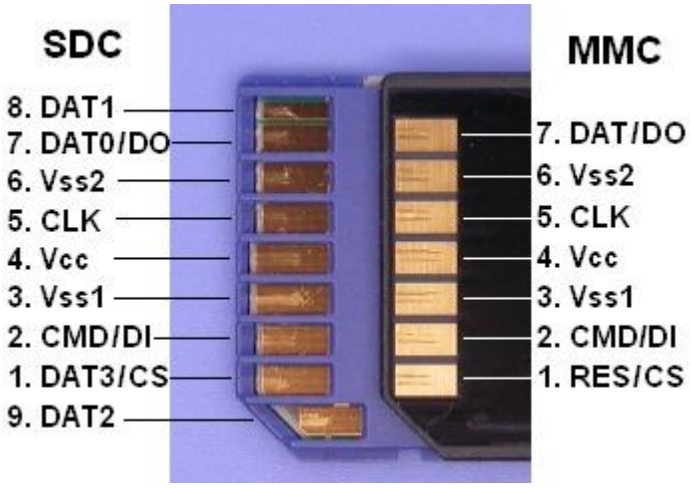

Figura 2.5: Esquema tarjeta SDC

La tarjeta MMC tiene 7 contactos y la SDC tiene 9 contactos, dos adicionales a las tarjetas MMC. La transferencia de datos entre el controlador y la tarjeta está dada por la transferencia serial de datos en base del reloj.

Los valores de voltaje de trabajo de las tarjetas esta dentro de los 2.7 a 3.6 volts. Mientras que la concepción de corriente puede ser superior a los 10 miliamperios para a cada caso, el controlador podría aplicar 100 miliamperios para su correcto funcionamiento.

El proceso de transmisión de datos está dado por la transmisión serial, por lo que nuestro controlador deberá trabajar en un modo SPI (*Serial Peripheral Interface)*. Las ventajas de una operación en modo SPI contempla el costo para aplicaciones embutidas. Para un modo de funcionamiento adecuado la frecuencia máxima de funcionamiento es de 0 a 20 o 25 Hz, pero para propósitos de inicialización de las tarjetas MMC la frecuencia de reloj debe estar limitada a 400khz. [1]

CAPÍTULO 3:

3. ELECTROCARDIOGRAMA (ECG)

Se presentará una explicación de la fisiología del corazón y una descripción detallada del tipo de señal biomédica con la que vamos a trabajar, para dar el detalle del circuito por nosotros adoptado como solución para el acondicionamiento de señales análogas de este tipo.

3.1 Fisiología del Corazón

En anatomía, el corazón es el órgano principal del aparato circulatorio. Es un músculo estriado hueco que actúa como una bomba aspirante e impelente, que aspira hacia las aurículas la sangre que circula por las venas, y la impulsa desde los ventrículos hacia las arterias.

Cada latido del corazón desencadena una secuencia de eventos llamados ciclos cardíacos, que consiste principalmente en tres etapas: sístole auricular, sístole ventricular y diástole. El ciclo cardíaco hace que el corazón alterne entre una contracción y una relajación aproximadamente 75 veces por minuto, es decir el ciclo cardíaco dura unos 0,8 segundos.

Durante la sístole auricular, las aurículas se contraen y proyectan la sangre hacia los ventrículos. Una vez que la sangre ha sido expulsada de las aurículas, las válvulas auriculo ventriculares entre las aurículas y los ventrículos se cierran. Esto evita el reflujo de sangre hacia las aurículas. El cierre de estás válvulas produce el sonido familiar del latido del corazón. Dura aproximadamente 0,1 s. [8]
La sístole ventricular implica la contracción de los ventrículos expulsando la sangre hacia el sistema circulatorio. Una vez que la sangre es expulsada, las dos válvulas sigmoideas, la

válvula pulmonar en la derecha y la válvula aórtica en la izquierda, se cierran. Durando aproximadamente. 0,3 s.

Por último la diástole es la relajación de todas las partes del corazón para permitir la llegada de nueva sangre. Durando aproximadamente. 0,4 s.

En el proceso se pueden escuchar dos golpecitos; el de las válvulas al cerrarse (mitral y tricúspide) y la apertura de la válvula sigmoidea aórtica.

El movimiento se hace unas 70 veces por minuto. La expulsión rítmica de la sangre provoca el pulso que se puede palpar en las arterias radiales, carótidas, femorales, etc.

Si se observa el tiempo de contracción y de relajación se verá que las aurículas están en reposo aproximadamente 0,7 s y los ventrículos unos 0,5 s. Eso quiere decir que el corazón pasa más tiempo en reposo que en el trabajo.[8]

3.2 Características de las señales del corazón para un electrocardiograma.

El corazón posee cuatro cámaras que funcionan como una bomba para el sistema circulatorio, este puede ser dividido en lado derecho e izquierdo, o a su vez bomba del lado derecho y bomba del lado izquierdo. Estas dos bombas poseen dos fases sístone (contracción) y diástone (relajamiento)[9]. La figura 3.1 muestra la anatomía del corazón.

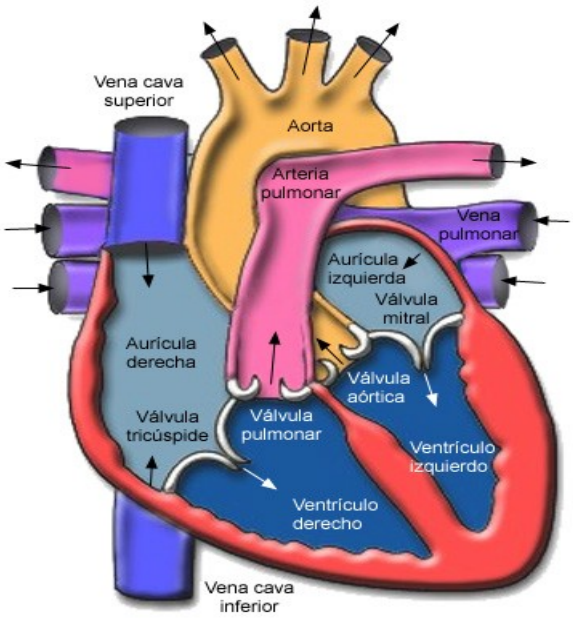

Figura 3.1: Anatomía del corazón

El lado derecho bombea la sangre para el ventrículo derecho que bombea la sangre para los pulmones a través de la arteria pulmonar.

Y el atrio izquierdo bombea la sangre para el ventrículo izquierdo. La contracción rítmica del corazón es precedida por una serie de actividades eléctricas bien coordinadas. Esta actividad eléctrica es intrínsica del corazón.

En el corazón existen marca pasos naturales y señales generadas por estos marca pasos, estas señales eléctricas son las que provocan contracción del miocardio (células del músculo cardíaco).

El nodo sino-areial (NSA): Marca pasos natural; los impulsos eléctricos son generados en este conjunto de células y transmitido de forma organizada para el atrio derecho y para el atrio izquierdo. El nodo atrio ventricular (NAV): Recibe los impulsos de NSA, y provoca un atraso y transmite a los ventrículos por medio de la membrana de *hiss*. La membrana de *hiss* con sus dos ramos y los pliegues subendocardios de fibras Purkinge conducen los impulsos eléctricos para los ventrículos [10]. Toda esta secuencia se muestra en la figura 3.2

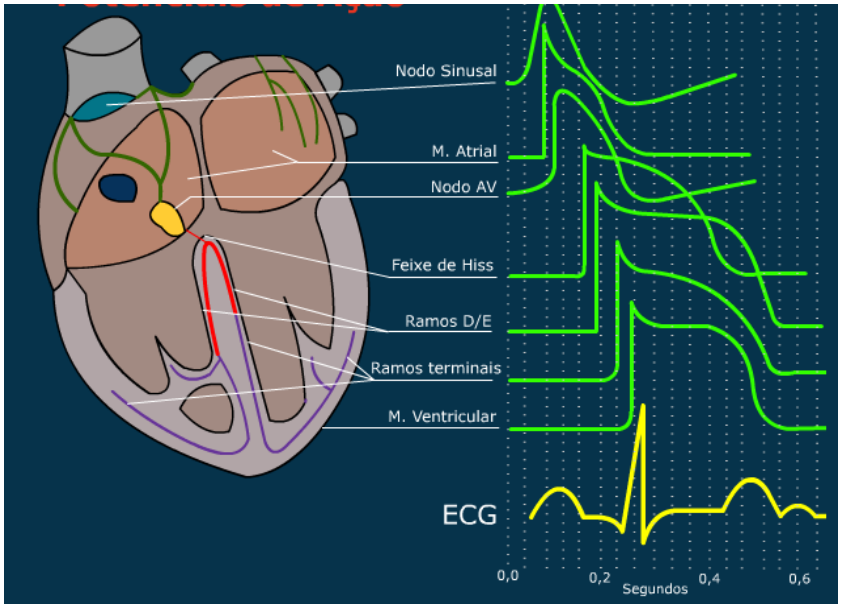

Figura 3.2: Señal ECG

La activación eléctrica del corazón lleva un flujo de corriente eléctrica en el tórax. Los potenciales medidos en la superficie del torax son llamados de electrocardiogramas o ECG. Aunque la activación eléctrica del corazón sea compleja el corazón puede ser visto como un generador eléctrico. Asumimos que para cada instante la actividad del corazón puede ser modelada por un dipolo eléctrico.

Un electrocardiograma (ECG) es una señal de respuesta a una técnica no invasiva con el propósito de realizar exploraciones en el sistema cardio vascular humano. Está compuesto de un conjunto de formas de onda que resultan de la des polarización y re polarización auricular y ventricular e indica la conducción de impulsos eléctricos a través del corazón, con el tiempo en abscisas y la tensión eléctrica en ordenadas, mediadas en mili segundos y mili voltios, respectivamente. Específicamente, en un ECG se reconocen tres componentes elementales: las ondas P, QRS y T [8]. Figura 3.3.

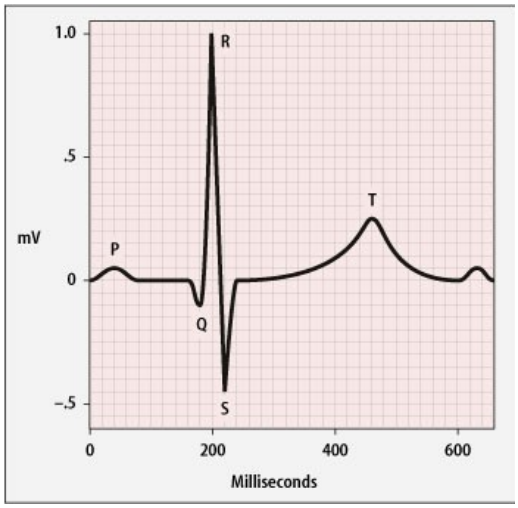

Figura 3.3: Ondas P, Q, R, S, T

La onda P representa el potencial eléctrico responsable por la des polarización del atrio que ocurre poco antes de su contracción. El conjunto QRS representa el potencial que se propaga por los ventrículos y que es responsable por la des polarización que también ocurre poco antes de la contracción. La onda T representa el potencial de polarización de los ventrículos, que ocurre cuando este está relajado.[11]

En algunos textos es también conocida al segmento final de la señal como forma de onda U, que siempre es detectada en una señal ECG y es generada por los potenciales tardíos del movimiento de dilatación del corazón y sus arterias.
La disposición de los electrodos en el cuerpo permite la obtención de diferentes amplitudes y formatos de ondas. En medicina fueron normalizadas posiciones de electrodos que constituyen un canal. En un ECG patrón son obtenidos doce canales, o sea, doce diferentes visiones del corazón.

En condiciones normales la duración media de las ondas P, conjunto QRS, e la onda T no exceden el 0,11s, 0,10s y 0,15s respectivamente y tienen amplitudes máximas de 0,3mV, 3mV, e 1mV respectivamente.

Par los canales que se utilizan de electrodos en los dos brazos y en una de las piernas la amplitud media de la onda P son alrededor de 0,1mV a 0,3mV y del conjunto QRS es en torno de 1mV y de la onda T entre 0.2mV y 0,3mV.

La frecuencia cardíaca de un ser humano varía de 30 a 220 latidos por minuto lo que equivale

a 0,5 a 3,67Hz.

Para detectar anormalidades, es a veces necesario disponer de registros de 24 horas. Si la tasa de muestreo es de 200 Hz y se utilizan 12 bits para describir cada una de ellas, la cantidad de espacio necesario se eleva a 26Mb. Para proceder a su almacenamiento y eventual transmisión, es necesario disponer de mecanismos de compresión de la señal que conserven la información relevante que facilite los diagnósticos posteriores.[11]

3.3 Características de un circuito acondicionador de una señal de electrocardiograma.

Antes de determinar las características de nuestro proyecto observemos cuáles son las características de ECG ya existentes.

- Circuito de protección: Es utilizado para tensiones muy altas en la entrada del ECG.
- Selector de derivación: Seleccionara cual derivación será utilizada. La selección puede ser manual o mediante el micro controlador.
  Este selector de derivación, es el que permitirá determinar la medida del potencial eléctrico entre un punto respecto a otro. Las derivaciones más comunes son las siguientes.

  DI: electrodo negativo en el brazo derecho, positivo en el brazo izquierdo.

  DII: electrodo positivo en el brazo derecho, negativo en el pierna izquierda.

  DIII: electrodo negativo en el brazo izquierdo, positivo en la pierna izquierda.

- Señal de calibración: Una señal de 1 mV es introducida en el ECG para verificar la calibración de tensión.
- Pre-amplificación: Es la primera amplificación de la señal. Esta sección debe tener alta impedancia y alto rechazo en modo común.
- Circuito de aislamiento: Funciona como una barrera contra el ruido de la red de 60 Hz.
- Circuito de pierna derecha: Funciona como referencia para evitar un aterramiento del paciente.
- Circuito amplificador: Amplifica la señal para que pueda ser adquirida adecuadamente por el conversor A/D.
- Microcontrolador.

Para este prototipo el circuito de adquisición de señales ECG tiene los siguientes requisitos: adquirir una señal analógica de baja amplitud, amplificar y filtrar, dando las condiciones adecuadas para su incorporación al conversor AD nuestro procesador ARM.

El sistema debe estar apto para recibir señales de tensión entre 0,1mV e 3,0mV combinando con una componente de DC de ±300mV (resultado del contacto piel-electrodo) y una tensión de modo común de 1,5 V (resultado de la diferencia de potencial entre los electrodos y la tierra).
La banda de frecuencia útil de una señal ECG varía de acuerdo con la aplicación, de 0,5Hz hasta 50Hz para unidades de monitoramientó UTI y sobre el 1Khz para medidas de potencial tardío (detección de marca-pasos). Un ECG clínico patrón tiene un ancho de banda de 0,05Hz a 100Hz. [7]

La señal ECG puede ser perturbada por diversos tipos de ruido. Las principales fuentes de ruido son:

1. Interferencia de la red eléctrica: 60Hz y armónicos de orden superior.

2. Ruido de contacto en los electrodos: la variación del contacto del electrodo con la piel causa un desbocamiento de la línea base de señal.

3. Contracción muscular: las señales típicas de electro miografia (EMG) son generadas y mezcladas con señales ECG.

4. Respiración: la respiración causa el deslocamiento de la línea de base de señal

5. Interferencia electromagnética: otros dispositivos eléctricos y electrónicos pueden causar interferencia con los cables conectados a los electrodos actuando como antenas.

La mayor fuente de interferencia para un monitor ECG son los sistemas de energía eléctrica. Además de proporcionar energía al electrocardiograma, las líneas eléctricas están conectadas suministrando a otros equipos típicos en un cuarto de hospital. Estas líneas de energía se encuentran en las paredes y en el cielo de un dormitorio y en otros puntos de un edificio. Estas líneas de energía pueden afectar la señal procesada por un ECG introduciendo interferencia en las señales trazadas. Este efecto lo veremos en las pruebas de nuestro ECG ya implementado su diferencia con una señal con ruido de 60Hz y sin esta interferencia. Los campos eléctricos entre las líneas de energía y las líneas del ECG, forman capacitancias parásitas las cuáles introducen esta interferencia, como se muestra en la figura 3.4. [7]

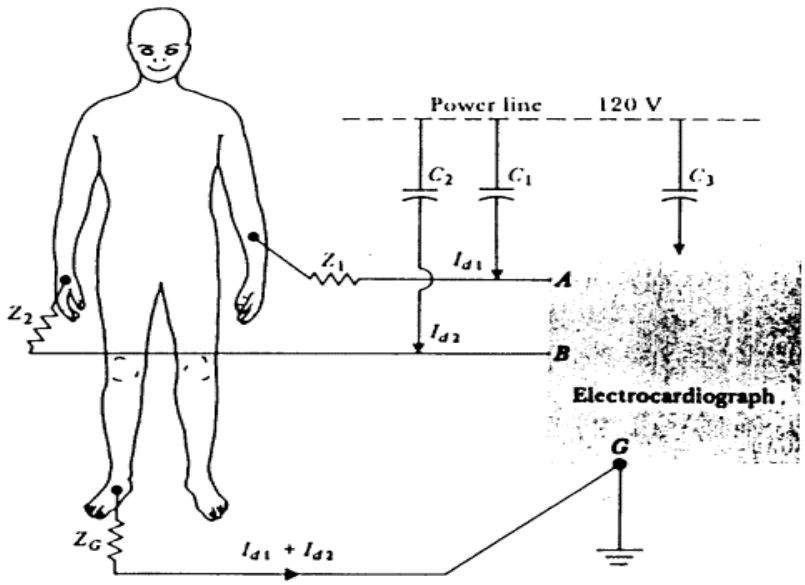

Figura 3.4: Capcaitancias parásitas al ECG

La segunda fuente de interferencia se ubica en el mismo paciente el cual forma unas capacitancias parásitas con las líneas de energía, como se muestra en la figura 3.5.

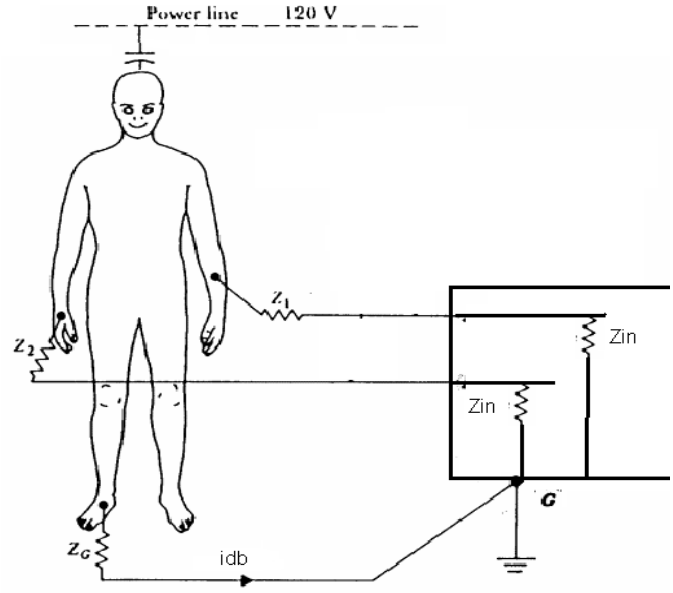

Figura 3.5: Capacitancias parásitas al paciente

Teniendo en mente la información podremos proyectar nuestro canal de adquisición de señal ECG.

Para nuestro prototipo se ha proyectado un circuito para adquisición de un canal de señal ECG. Compuesto por cinco módulos, de modo a minimizar el efecto de los ruidos: un módulo amplificador diferencial, un módulo filtro pasa-altas, un módulo filtro pasa-bajas asociado a un amplificador buffer, un módulo para generar la referencia de la pierna derecha y un módulo para generar las tensiones de amplificación y tierra común para alimentación de los demás módulos. El esquema de conexión entre los módulos deberá seguir el diagrama mostrado en la figura 3.6

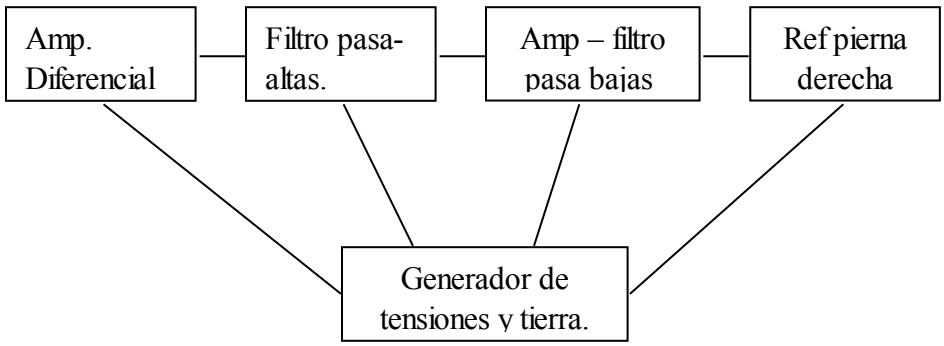

Figura 3.6: Diagrama de conexión

Los elementos que requeriremos para desarrollar nuestro canal de adquisición se compondrán básicamente por un amplificador de instrumentación el cual lo utilizaremos para la elaboración de nuestro amplificador diferencial y varios amplificadores operacionales que se utilizaran para la construcción de los filtros pasa-bajas, pasa-altas y el circuito de referencia de la pierna derecha. Estos requerimientos deberán tener las siguientes características técnicas:

Amplificador de Instrumentación:

1. Ganancia estable (G = 1 a 10).
2. Alto rechazo de modo común.
3. Baja corriente de entrada *bias* (I_B).
4. Buen balance de vías de salida.
6. Bajo *offset* y *drift. [7]*

Amplificador Operacional

- Bajo ruido y alta ganancia (G = 1 a 1000).
- Salida *Rail-to-Rail.*
- Bajo *offset* y *drift.[7]*

Después de una indagación de los componentes que más se acercan a las necesidades planteadas llegamos a la conclusión que los dispositivos más apropiados son los siguientes:

INA118

Tipo: Amplificador de instrumentación.
Fabricante: *Texas Instrument*.
Rechazo en modo común: 110-db.
Ganancia: 100.
Offset: max 110-μV.
Drift: 0.4- μV/ºC.
I_B : 2nA.
Empaquetamiento: DIP-SO8.

TL064

Tipo: Amplificador Operacional.
Fabricante: *Texas Instrument.*
Offset: max 5-μV.
Drift: 0.05- μV/ºC.
Empaquetamiento: DIP – SO8.

Para el circuito de referencia de tierra, utilizaremos el TLA2435, el cual es un divisor de tensión para generar nuestra tierra virtual, este dispositivo posee un empaquetamiento de tipo LP, y es ideal ya que impide una entrada adicional de interferencia.

3.3.1 Pre-amplificador.

La etapa de pre-amplificación consta del amplificador de instrumentación, en este caso utilizamos el INA118. Esta etapa consta de dos partes la primera es la que obtiene una señal del brazo izquierdo y derecho del paciente y la segunda que es nuestro conocido circuito de pierna derecha, el cual sirve para cerrar el circuito en el paciente. El propósito del primer circuito es el de crear un circuito de alta impedancia de entrada y buen rechazo en modo común. El circuito es el siguiente como muestra el esquema de la figura 3.7.[10]

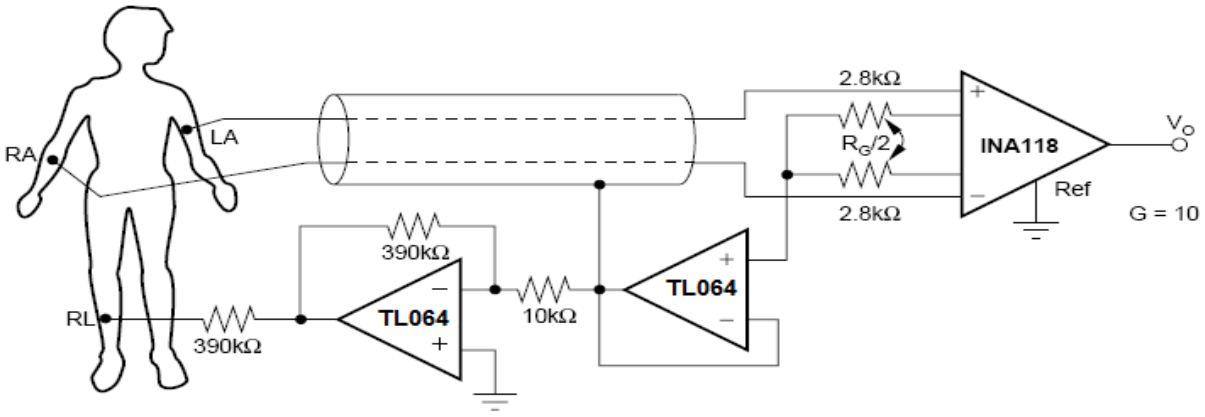

Figura 3.7: Pre amplificador.

3.3.2 Etapa Filtro pasa-altas

El propósito de esta etapa es el de remover la componente de DC. La frecuencia de corte deberá ser de 0.1 Hz. Nuestro circuito simplemente es un filtro pasivo como muestra en el esquema de la figura 3.8.

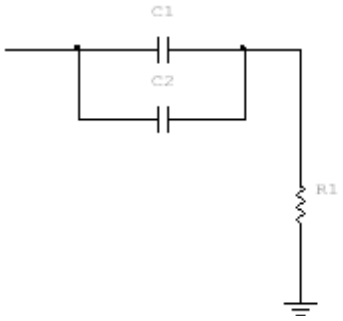

Figura 3.8: Filtro pasa-altas

3.3.3 Etapa Buffer.

Nuestro tercer circuito és un *buffer* este debe presentar una ganancia de 100 V/V, para obtener una ganancia total de 1000 V/V. El amplificador que se desea implementar es un amplificador no inversor, como muestra el esquema de la figura 3.9.

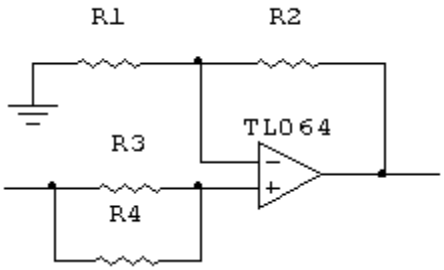

Figura 3.9: Buffer

3.3.4 Etapa Filtro Pasa-bajas.

Este filtro pasa bajas debe ser proyectado basándose en una topología *sallen-key*, este circuito será un filtro *butterworth* el cual debe poseer dos polos, a ser diseñado para una frecuencia de corte de 100 Hz el filtro, se muestra en el esquema de la figura 3.10.

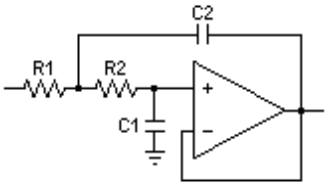

Figura 3.10: Filtro Pasa-bajas.

3.3.5 Etapa filtro Notch 60 Hz

Debido a la alta interferencia que posee nuestro circuito, se vio en la necesidad de implementar una etapa adicional a la planificada, esta es un filtro *Notch* para una frecuencia de 60 Hz. La señal de ECG tiene la característica que posee componente superior a 60 Hz e inferior a esta inclusive posee componente de 60 Hz; por lo que un filtro pasa bajas en 60 Hz seria una decisión errada, para efecto de prototipo se decisión implementar este filtro esperando que nuestra señal no termine completamente distorsionada. El filtro a implementar contempla el siguiente esquema, como se muestra en el esquema de la figura 3.11. [6]

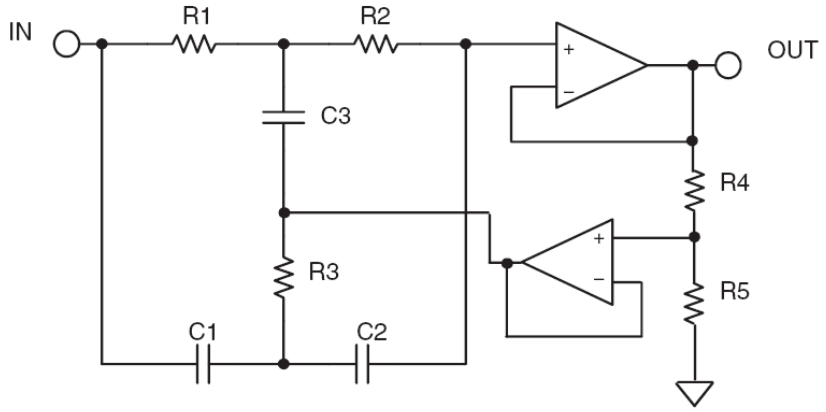

Figura 3.11: Filtro Notch

Una característica muy importante del filtro *noch* es que posee una respuesta infinita negativa para la frecuencia a diseñar de manera teórica, como se muestra en el esquema de la figura 3.12. [11]

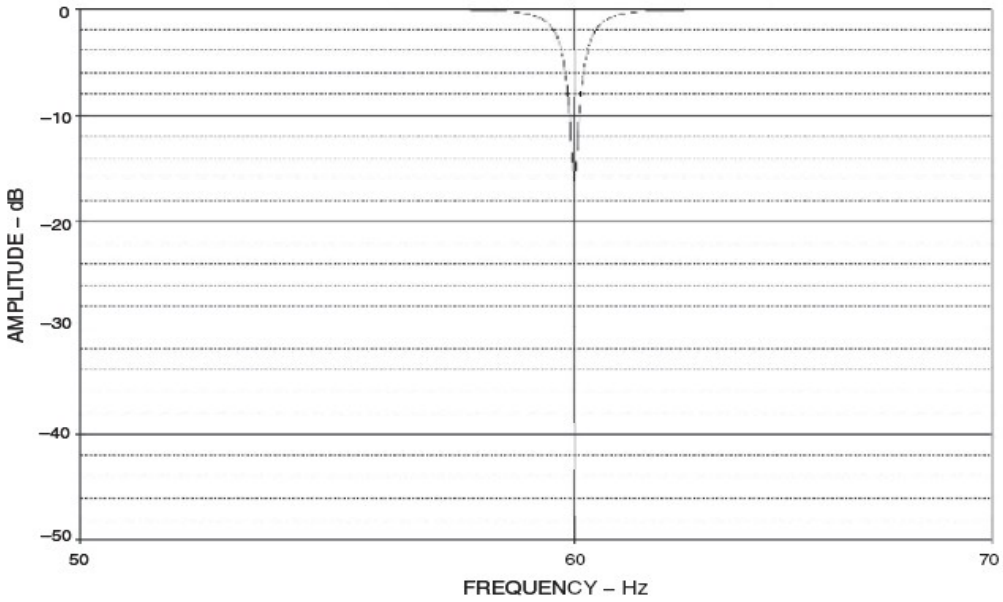

Figura 3.12: Filtro Notch 60 Hz

CAPÍTULO 4:

4. CIRCUITO ACONDICIONADOR.

Se presentarán las características funcionales de conversión AD y los motivos para el empleo de uno específico. Se dará una explicación de su utilización como solución para digitalización de señales ECG. Se incluirá también la descripción de circuitos adicionales utilizador en el proyecto

4.1 Características del Conversor AD.

El procesador escogido tiene la característica de tener embebido un conversor analógico digital de 10 bits, el cual se basa en un Registro de Aproximaciones Sucesivas (SAR, *Successive Approximation Register).* Posee 8 canales, de los cuáles 4 se encuentran multiplexados, lo cual permite la conversión de 8 señales en línea y su rango será de 0v hasta el voltaje de referencia ADVREF. Adicional tiene la característica de disparar una señal de *trigger* por medio software, la de aceptar una señal externa de *trigger* y la de administrar señales de *trigger* por medio de temporizadores y contadores. Además son configurables la sincronización del conversor, el tiempo de *startup,* y el tiempo de muestreo.

El poseer un dispositivo embebido de conversión dentro de nuestro procesador mejora el rendimiento del prototipo, ya que basándose en experiencias anteriores donde los efectos del ruido se incrementaron con el uso proporcional a componentes externos al procesador, se determino que la solución adecuada es la utilización de dispositivos embebidos. El punto desfavorable se basa que la resolución de nuestro conversor es apenas de 10 bits y las normas médicas para este tipo de equipos piden un mínimo de 12 bits, pero como nuestro prototipo se encuentra en etapa de desarrollo, en un futuro podrá ser remplazado por un procesador de la misma familia el cual tenga de manera embebida un conversor de mayor resolución. Aceptaremos el conversor actual para realizar experiencias y determinar si la resolución que

estamos utilizando es la adecuada para alguna aplicación en las que no sea indispensable una resolución de 12 bits, este tipo de experiencias nos ayudarán a tener una visión del tipo de información que se manejará en un futuro y a partir de estas pruebas experimentales proyectar dispositivos de almacenamiento adecuados.

4.2 Descripción de circuitos adicionales.

Según la información expuesta en el capítulo 3, se ha determinado las características de las diferentes etapas de nuestro circuito de adquisición.

4.2.1 Módulo Amplificador Diferencial.

Para el amplificador diferencial fue escogido el circuito integrado INA118 de la *Texas Instrument*. Este dispositivo es un amplificador de instrumentación con ganancia regulable que posee alta sensibilidad a señales bio-médicas, alta precisión, alto valor de rechazo en modo común CMRR (*Relación de rechazo en modo común*) y baja potencia de funcionamiento.

La entrada para este módulo esta compuesta por los electrodos que son colocados en los brazos u hombros de la persona analizada. Según la hoja técnica de este dispositivo la ganancia es ajustada a partir de un resistor colocado entre sus pines 1 y 8 como se muestra en la figura 4.1. La ecuación para el cálculo de ganancia está dada por la ecuación 4.1.

$$G = 1 + \frac{50k\Omega}{R_G} \quad \text{(Ec 4.1)}$$

Donde G es la ganancia obtenida y R_G es el resistor que determina la ganancia.

Para este módulo proyectamos una ganancia de 10v/v. La ecuación 4.2 muestra el valor necesario del resistor.

$$10 = 1 + \frac{50x10^3}{R_G} \therefore R_G = \frac{50x10^3}{10 - 1} \therefore R_G = 5{,}56k\Omega \quad \text{(Ec 4.2)}$$

Como el valor obtenido para R_G no es comercial y como necesitamos de una tensión para ser

utilizada en el módulo de referencia de la pierna derecha cuyo valor es la mitad de la tensión entre los terminales 1 y 8 del dispositivo, fueron escogidos dos resistores de 2,70kΩ cada uno. Ese valor fue escogido por ser un valor comercial cuyo montaje en serie resulta el valor de 5,40kΩ que el valor que mas se aproxima al valor teórico de $R_{G,}$ el valor de ganancia real y teórica con estos valores se los muestra en la tabla 4.1 y 4.2. [12]

La figura 4.1 muestra la configuración del módulo Amplificador Diferencial

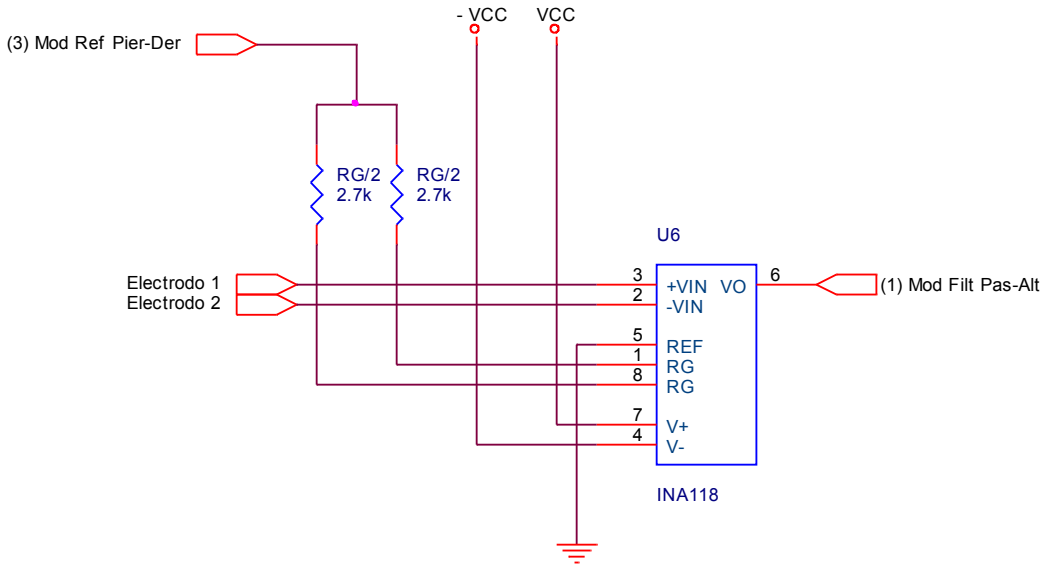

Figura 4.1: Módulo Amplificador Diferencial

Los resistores que se utilizaron fueron de tolerancia del 1%, con el objetivo de obtener mayor precisión en la implementación de nuestros circuitos. Se realizo mediciones en los elementos utilizados para realizar un cálculo real de la respuesta que obtendríamos, adicionalmente mediciones de las respuestas obtenidas, los resultados se encuentran tabulados en las tablas 4.1 y en la tabla 4.2

	Valor Teórico KΩ	Valor Real KΩ
R1	390	381
R2	390	385
R3	10	9.94
R4	2.8	2.691
R5	2.8	2.69
G	10	**10.29195317**

Tabla 4.1: módulo Amplificador Diferencial

Vin mV	0.2
Vout V	1.92
G real	**9.6**

Tabla 4.2: módulo Amplificador Diferencial Ganancia real.

4.2.2 Módulo Filtro Pasa-altas.

El propósito de esta etapa es eliminar la componente de DC, para lo cual la frecuencia de corte deberá ser muy baja; el diseño de un filtro activo para este valores de frecuencias se complica demasiado los valores de los componente se consideran inadecuados para una aplicación, por lo que una decisión adecuada es la de implementar un filtro pasivo; debemos considerar que los capacitores deberán ser cerámicos y no electrolíticos, ya que nuestra señal tendrá una polaridad positiva y negativa. [12]

La frecuencia de corte deseada para este filtro es de 0,1Hz para retirar las componentes de DC generadas por el contacto de los electrodos con la piel. Por tanto se proyecto un filtro RC de primer orden.
La entrada de ese módulo filtro pasa-altas es una salida del módulo amplificador diferencial.

La relación entre los valores de resistencia, capacitancías y frecuencias de corte es dada por la ecuación 4.3.

$$\omega_c = \frac{1}{RC}, \omega_c = 2\pi f_c \therefore \; f_c = \frac{1}{2\pi RC} \quad \text{(Ec 4.3)}$$

Donde w_c es la frecuencia de corte, R es el resistor, y C es el capacitor.

El valor del capacitor siempre será más limitado en el mercado de componentes, por lo que lo fijaremos desde un comienzo. Partiendo del valor comercial del capacitor disponible de 1µF y mediante una estructura en paralelo de tres componentes y con la frecuencia de 0,1Hz tenemos el valor del resistor necesario dado por la ecuación 4.3, el cual es de 820 KΩ
Para el diseño de este filtro se debe considerar un criterio muy importante el cual al existir una impedancia muy alta de este como entrada para la próxima etapa, limitara la corriente pudiendo ser insuficiente para entrar en el rango de funcionamiento por lo que es importante que no sobrepase el mega ohm. La disposición de circuito se muestra en la figura 4.2. El diseño de nuestro circuito cumple con esta condición ya que el resistor que va a tierra apenas es de 820 KΩ.

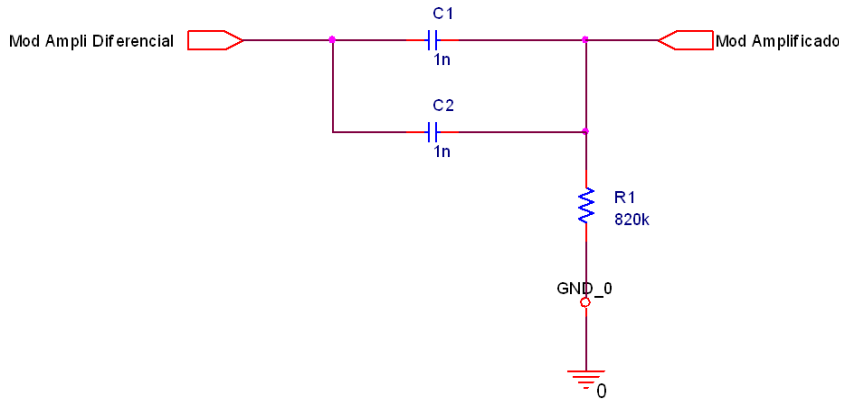

Figura 4.2: Filtro pasa-altas

Se encuentra en la tabla 4.3 los valores de los componentes y el cálculo de la frecuencia de corte esperada y real. De nuestro circuito.

	Valor Teórico	Valor Real
R1 KΩ	820	827
C1 µF	2	2,022
Fc	0,097Hz	0,095Hz

Tabla 4.3: Filtro pasa-altas

4.2.3 Módulo Amplificador.

El módulo amplificador diferencial posee una ganancia de 10v/v. Como necesitamos de una ganancia total de 1000v/v para este módulo fue proyectada una ganancia de 100v/v. El *buffer* que se implemento es un amplificador no inversor con una ganancia de 101 v/v debido a los componentes utilizados, se acepto este valor de ganancia debido a que en la etapa anterior la ganancia no fue real a la proyectada, por tanto con esta ganancia superior se espera obtener una compensación.

El circuito implementado me muestra en la figura 4.3.

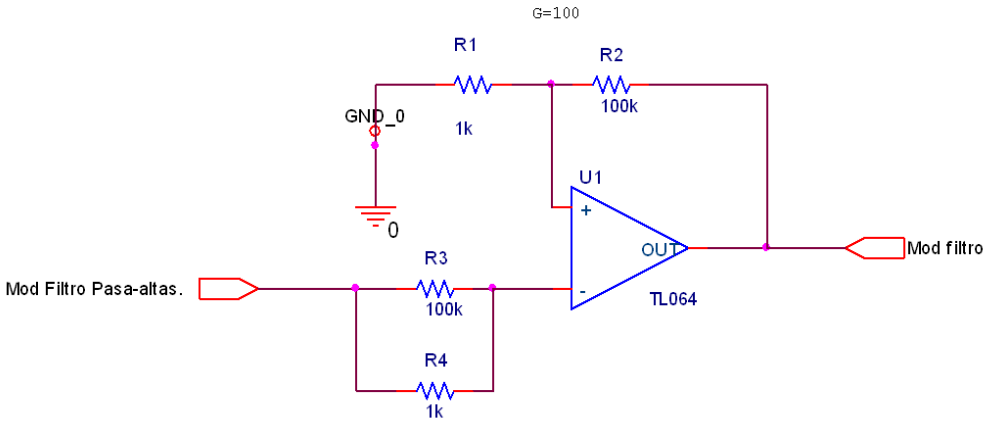

Figura 4.3: Buffer con ganancia

De idéntica forma que en etapas anteriores se realizo la medición de los valores de los componentes y se obtuvo los siguientes resultados, como se muestra en la tabla 4.4.

	Valor Teórico	Valor Real
R1 KΩ	1	0.995
R2 KΩ	100	99.6
R3 KΩ	100	99.5
R4 KΩ	1	0.996
G V/V	101	**101**

Tabla 4.4: Buffer con ganancia

A rigor se realizaron pruebas en las cuáles se midió la salida de las señales comparando con los resultados teóricos del mismo, como se muestra en la figura 4.5.

Vin real	32.8
Vout real	3.08
G med	93.90243902

Tabla 4.5: Buffer con ganancia

4.2.4 Modulo filtro pasa-bajas.

Nuestro filtro pasa bajas fue proyectado en la topología *sallen – key,* l característica de este filtro es la de poseer dos polos, y ser diseñado para una frecuencia de corte de 100hz. El resultado del diseño es el filtro se muestra en la figura 4.4.

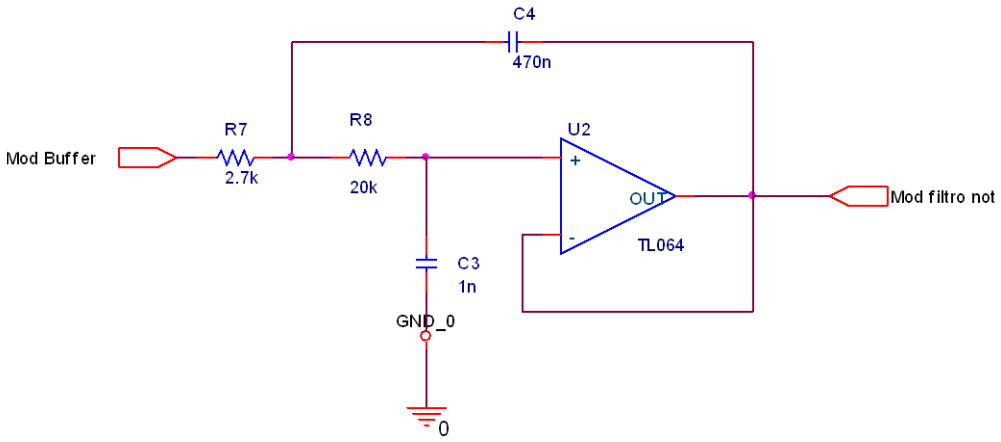

Figura 4.4: Filtro pasa-bajas

El filtro se diseña para una frecuencia base de 100 hz, la cual bien caracterizada por la ecuación 4.4

$$f_c = \frac{1}{2\pi\sqrt{R_1 R_2 C_1 C_2}} \quad \text{(Ec 4.4)}$$

Donde f_c es la frecuencia de corte y R_1 , R_2, C_1, C_2 son los resistores y capacitores que se demuestran para la topología utilizada, como se mostró en la figura 4.4.

Los componentes tienen los siguientes valores para los cuáles fue calculada la frecuencia deseada. Como se muestra en la tabla 4.6.

	Valor Teórico	Valor Real
R1 KΩ	2.7	2.69
R2 KΩ	20	20.03
C1 nF	100	98.8
C2 nF	470	440

Tabla 4.6: Filtro pasa-bajas

Con estos valores de componentes se realizo en cálculo de la frecuencia de corte, y se medio con ayuda de un osciloscopio la frecuencia de corte real, como se muestran en la tabla 4.7.

fc calc	99.90	103.99
fc med	103.50	

Tabla 4.7: Filtro pasa-bajas

4.2.5 Filtro Notch 60 Hz.

El filtro notch que se implemento tiene el siguiente esquema, como se muestra en la figura 4.5.

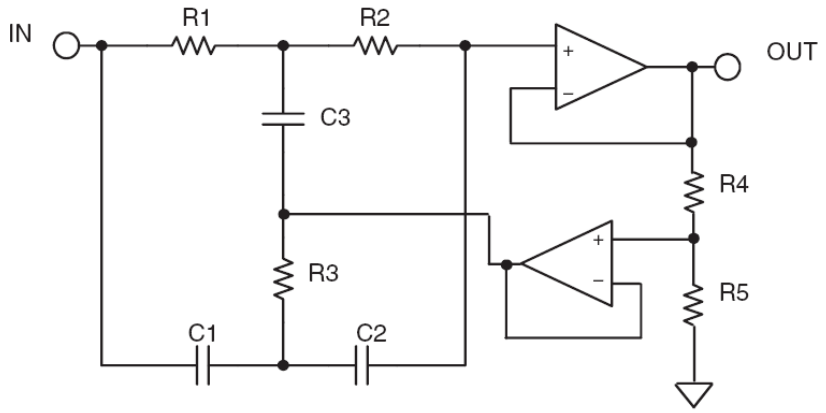

Figura 4.5: Topología Filtro Notch

Este filtro se ve caracterizado por las siguientes ecuaciones, las cuáles permiten determinar los componentes requeridos para su implementación.

Este modelo de filtro se encuentra caracterizado por el valor *k,* tal valor se encuentra descrito por la ecuación 4.5

$$k = 2\pi F_o C \qquad \text{(Ec 4.5)}$$

Donde F_o es la frecuencia de aplicación del filtro.
Este valor de calidad *k* debe cumplir como condición la ecuación 4.6.

$$R = \frac{1}{k} \qquad \text{(Ec 4.6)}$$

Los valores de resistores y capacitores del circuito se encuentra caracterizada por la ecuación 4.7, y correspondientes a la figura 4.5.

$$R = R1 = R2 = 2R3$$
$$C = C1 = C2 = \frac{C3}{2} \qquad \text{(Ec 4.7)}$$

La frecuencia de corte F_o es caracterizada por la ecuación 4.8.

$$F_0 = \frac{1}{2\pi RC} \qquad \text{(Ec 4.8)}$$

Donde *R* es el valor del resistor y *C* es el capacitor los dos caracterizados por la ecuación 4.7

Según la figura 4.5 observamos que los resistores *R4* y *R5* son adecuados para realimentar al circuito, para tal propósito se debe escoger un valor adecuado de *R´,* que cumpla con las ecuaciones 4.9 y 4.10, tal factor *K* cumplirá con la ecuación 4.11, en donde el valor Q es llamado coeficiente de correspondencia.

$$R4 = (1 - K)R' \qquad \text{(Ec 4.9)}$$

$$R5 = KR' \qquad \text{(Ec 4.10)}$$

Para un valor de *K = 1;* se eliminaran los valores de *R4* y *R5,* donde *R5* va a tender a cero y *Q*

a infinito.

$$K = 1 - \frac{1}{4Q} \qquad \text{(Ec 4.11)}$$

Se debe recordar que para obtener un resultado adecuado, el factor de calidad Q debe ser proyectado para que los componentes sean accesibles en el mercado.
Basándonos en los resultados de nuestro filtro los valores de los componentes fueron los siguientes como se muestra en la tabla 4.8.

C1 nF	227
C2 nF	225
C3=C//C nF	424
R1=R+R KΩ	12.34
R2=R+R KΩ	12.34
R3 KΩ	6.76

Tabla 4.8: Filtro Notch

Como resultado el filtro que se implemento fue de la siguiente forma, como se muestra en la figura 4.6.

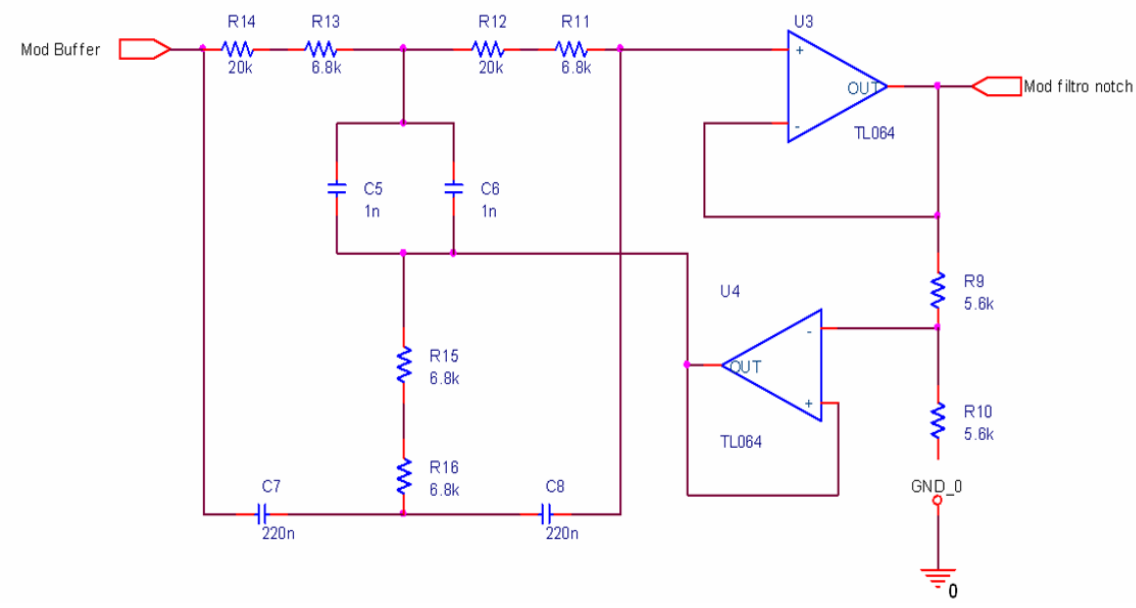

Figura 4.6: Filtro Notch

En las pruebas de laboratorio se encontró que la frecuencia de corte no es en los 60Hz deseados la frecuencia de corte fue en 59,02Hz muy próximo al valor diseñado. Como la respuesta de nuestro filtro tiene una caída infinita en la frecuencia de corte existente esta se abrirá en la parte superior como se muestra en la figura 4.7, eliminando gran parte de la componente de 60 Hz, lo cual para este prototipo experimental es muy adecuado ya que elimina gran parte de la componente de 60 Hz, observándose que la salida resulta una señal muy limpia.

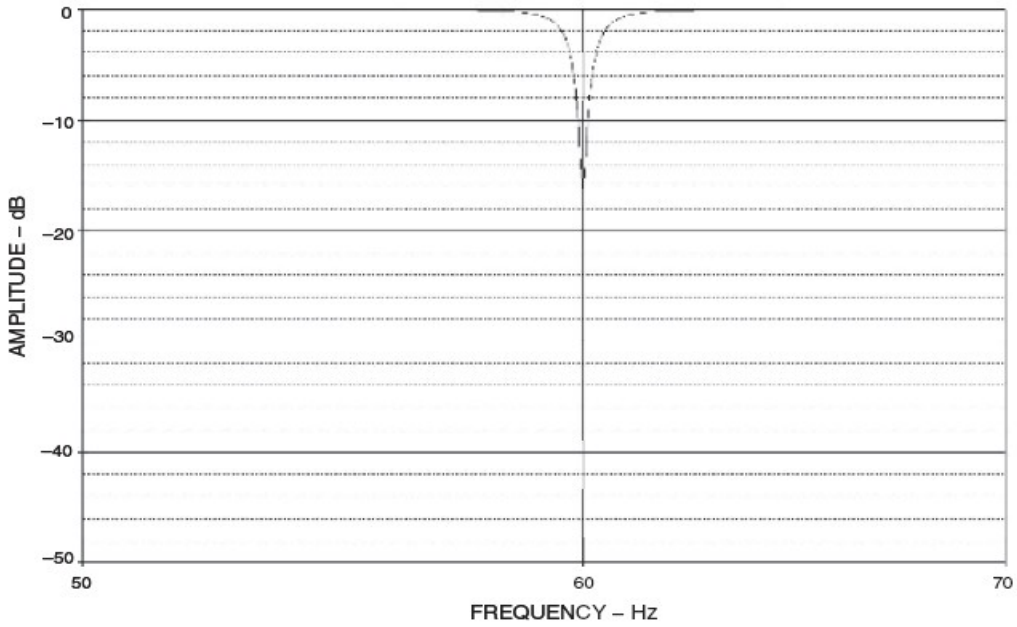

Figura 4.7: Respuesta Filtro Notch

4.2.6 Módulo de Referencia de Pierna Derecha.

Para minimizar la interferencia de 60Hz y mejorar la calidad de la señal en la salida del sistema fue adoptada la utilización del circuito de referencia de pierna derecha. Ese circuito consiste de una realimentación entre paciente y electrodo donde pocos microamperios son conducidos por el cuerpo del paciente en la pierna derecha hasta los electrodos en los brazos. Esa realimentación coloca en niveles bajos el ruido de modo común a través del aumento CMRR resultante. [10]

Un inconveniente de esa técnica es la posibilidad de oscilación si caso ocurra una mudanza de fase en la señal que ocurre en el cuerpo humano. Ese caso indeseado ocurre se hace presente a

la existencia de altas frecuencias de oscilación en la señal ECG.

En este módulo la entrada positiva de la primera etapa amplificadora es conectada a la mitad de la tensión entre los terminales que determinan la ganancia en el módulo amplificador diferencial. En el caso para este trabajo la señal fue conectada en el punto de unión de dos resistores de 2,7KΩ del módulo amplificador diferencial, como se muestra en la figura 4.8.
La primera etapa es un circuito seguidor de tensión. La salida de este seguidor esta conectada a una segunda etapa amplificadora en la configuración inversora con ganancia de 39 V/V e con la corriente de salida limitada por un resistor de 390KΩ. [9]

El dispositivo escogido para ese módulo fue un amplificador TL064 de la *Texas Instrument* que posee altísima precisión, bajo consumo de potencia e considerable anchura de banda.
La salida de este módulo esta conectada al electrodo colocado en la pierna derecha o en la parte inferior derecha del estomago de la persona a ser examinada.

La disposición de nuestro módulo de referencia de Pierna Derecha se encuentra diagramado en la figura 4.8.

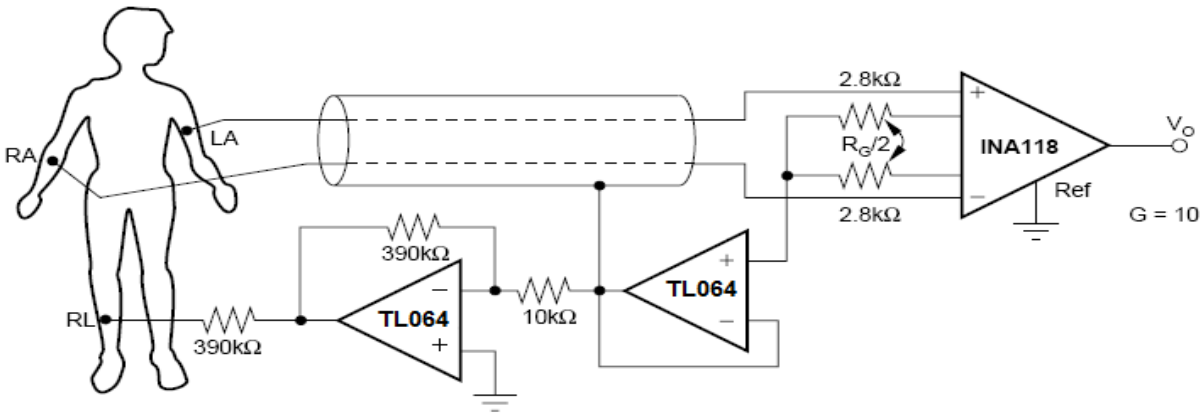

Figura 4.8: Referencia Pierna derecha.

4.2.7 Módulo Generador de Tensiones de alimentación y Tierra.

En este trabajo la pretensión de aprovechar la alimentación de 5v provista por el computador a través del cable USB, obligo a proyectar un circuito para alimentar a los amplificadores de los demás módulos, y para generar una tierra virtual del circuito. Para permitir una tensión negativa y otra positiva a partir de +5v y 0v se utilizo el TLE2425 el cual realizara un divisor de tensión que permitirá la doble polaridad y adicionalmente nos proporcionara una tierra virtual la cual la necesitamos para tener una referencia para nuestro circuito.

La figura 4.9 muestra un diagrama total de cada etapa del canal de adquisición analógica.

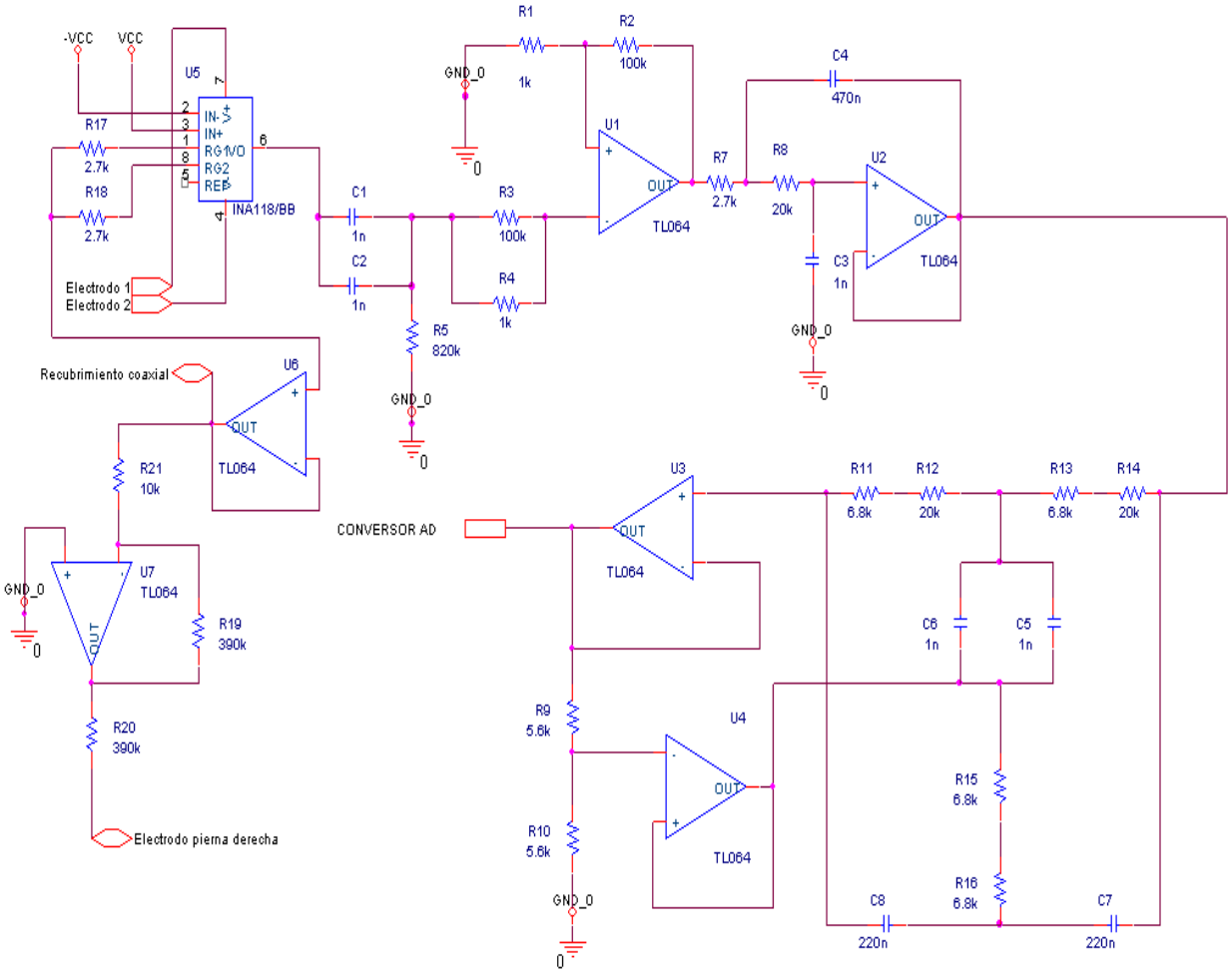

Figura 4.9: Diagrama canal analógico de adquisición

Los resultados producto de la implementación de nuestro canal analógico se muestra en la figura 4.10, la cual muestra a la señal ECG, sin el filtro notch

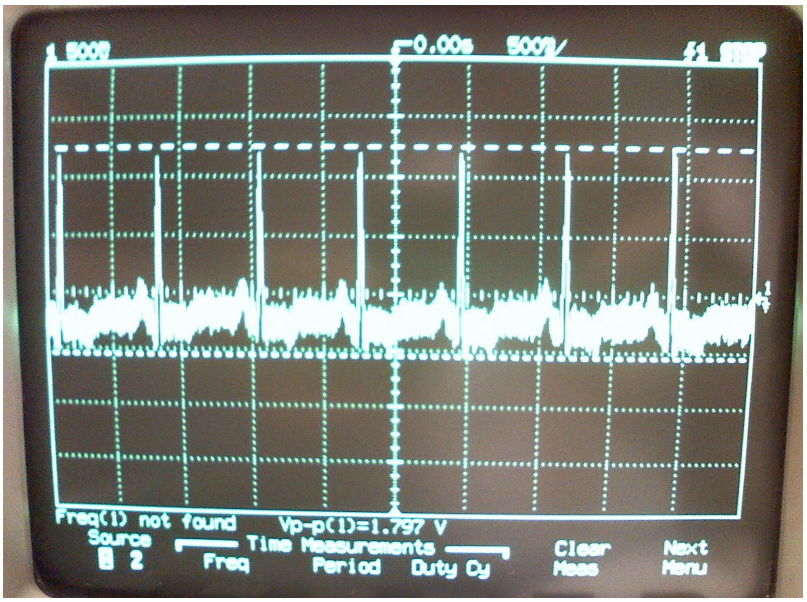

Figura 4.10: Señal ECG sin filtro notch.

La figura 4.11 muestra la señal ECG con la implementación del filtro notch, podemos notar una diferencia apreciable en la cantidad de ruido existente.

Figura 4.11: Señal ECG con filtro notch.

CAPÍTULO 5:

5.1 PROGRAMAS.

Se dará el detalle del firmware desarrollado para los procesos de adquisición, conversión, almacenamiento, transmisión y recepción de datos.

5.1.1 Procesos de enumeración.

El firmware desarrollado para el micro controlador esta dividido en 4 bloques: Laso principal, laso secundario del Temporizador, laso secundario del conversor Analógico/Digital, y laso secundario de transmisión del dispositivo USB, en la figura 5.1 se muestra como se relaciona cada bloque, todo este proceso se lo realiza conjuntamente con el proceso de enumeración del dispositivo USB el cual se encuentra en laso principal. El laso principal se lo utiliza para inicializar todos los periféricos utilizados en la placa, mientras que los lazos secundarios son utilizados para configurar los subdispositivos utilizados como son las entradas a los pines del ADC o salidas para el controlador *sample and holder* y los indicadores *led.*

El proceso de enumeración del dispositivo USB se dará de tal forma que una vez energizado el dispositivo mediante el cable USB realice las tentativas necesarias para que el dispositivo se encuentre reconocido para su funcionamiento e interacción con el sistema operativo.

El programa entra a un laso infinito donde verifica si el dispositivo UDP (*USB Device Port*) se encuentra configurado. Después de la verificación, en el caso del registro FIFO (*First in First out*) se encuentre completamente lleno, el programa informa la situación para que el dispositivo UDP realice la transferencia del paquete. El diagrama del flujo de tal proceso se encuentra especificado en la figura 5.1.

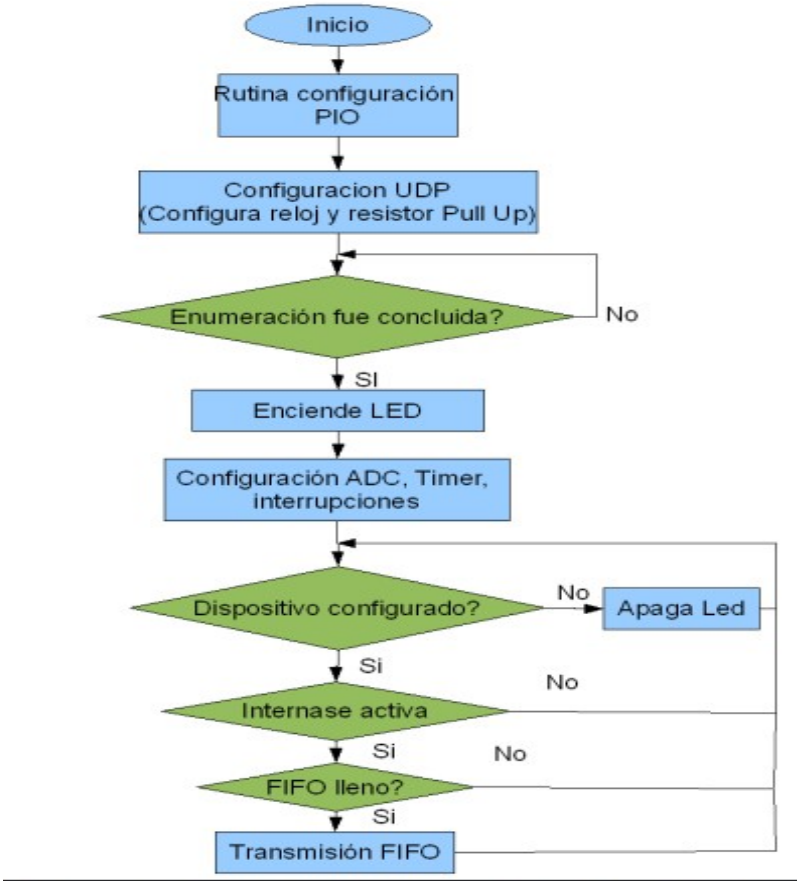

Figura 5.1: Proceso de enumeración e inicialización

Este proceso conceptualmente debería entregar una tasa máxima de transferencia debido a que la frecuencia de muestreo es mucho menor respecto a la velocidad de transferencia.

Una vez que implementado todo el firmware se realizaron pruebas adecuadas con un solo canal de adquisición, obteniendo resultados adecuados a las expectativas del prototipo, pero con la premisa de incrementar canales de digitalización para observar su desempeño. Una ves que fueron configurados canales adicionales de digitalización se pudo observar que la transferencia de datos al PC fue realizada de manera ágil pero no adecuada, ya que era notorio la pérdida de datos al utilizar más canales 4 canales de conversión. Debido a esta razón el proyecto final se lo dejo con un solo canal, pero con la premisa de solucionar este inconveniente teniendo en mente que ECGs comerciales utilizan como mínimo 8 canales de adquisición. Como tentativa se empezó a reducir el código y las rutinas, utilizando un mínimo de comparaciones y manteniendo apenas los bloques necesarios. Aunque persistiendo el mismo problema. Se coloco una señal del micro controlador para que nos muestre con un

cambio lógico en el momento que inicia la transferencia y en el momento que termina, para de esta manera poder medir la duración de la transferencia con ayuda de un osciloscopio. El resultado se muestra en la figura 5.2.

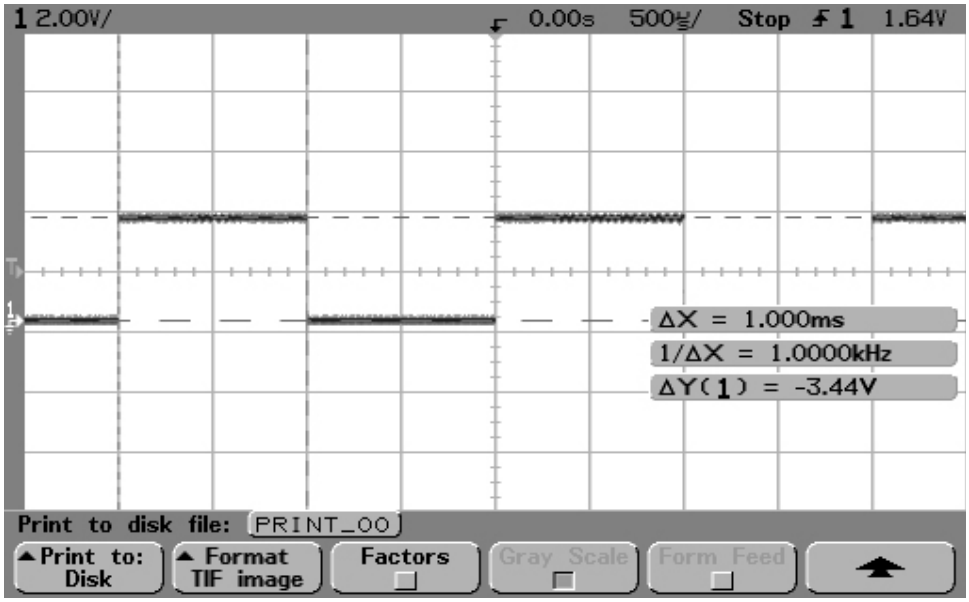

Figura 5.2: Tiempo de transferencia.

Como se observa, cada transferencia lleva aproximadamente 1 ms. Como el tamaño máximo de cada *endpoint* es de 64 bytes, tenemos una tasa de 64 kbps, lo que coincide con los resultados al utilizar 4 canales del conversor ADC. Como práctica se incremento el número de canales de conversión, encontrándose que la información perdida se incremento de manera progresiva. Se pensó en una posibilidad para este caso, la cual se basa en que los *drivers* para los dispositivos genéricos USB del tipo HID en el Windows no cubren con totalidad sus funciones.

La rutina de enumeración es llamada cuando se verifica una transferencia de control, realizada a través del *endpoint*. Esta identifica el tipo de pedido hecho por el *host*, interpretando los datos, y respondiendo adecuadamente.

El proceso de enumeración se da mediante los siguientes pasos:

1. La función inicia en estado desconectado, al ser insertada en un conector *Downstream*, se coloca en un estado llamado de alimentación.
2. El *hub* raíz detecta electrónicamente (Mudanzas de estado en D+/D- de SE0 para J) y actualiza su registrador STATUS_CHANGE, caracterizando el inicio del proceso de

enumeración.
3. El controlador *Host* inicia una serie de mensajes del tipo PC *host Request*, direccionados para el *hub* y para la función

5.1.2 Proceso de adquisición.

Para realizar una adquisición de datos adecuada se configuro el temporizador de tal forma que mediante interrupciones adecuadas a la frecuencia de muestreo, active al conversor analógico/digital, ya que nuestro canal analógico tiene una banda de 100 Hz, se determino una frecuencia de muestreo de 400 Hz de esa manera cumpliendo la ley Nyquist para la digitalización de una señal analógica.

La función de inicialización del temporizador, es llamada por la rutina principal, habilita el canal cero del temporizador con la siguiente configuración.

- El reloj seleccionado como 1/8 del reloj de la placa (MCK/8)
- Modo de forma de onda (WAVEFORM)
- El contador se reinicia cuando el valor de RC llega al especificado (WAVSEL=10b)
- Interrupción generada al reinicio del contador [3]

Como el reloj principal de la placa es de 48 Mhz, tenemos que el contador será incrementado a una frecuencia de 6 Mhz (a un periodo de 0,1667 us). Ajustándose al valor de RC para 749 tenemos aproximadamente un periodo de 0,125 ms que por tanto las interrupciones serán generadas a una frecuencia de 8 khz. La rutina de tratamiento de interrupciones apenas limpia la bandera que genera la interrupción y verifica si el ADC esta en medio de un grupo de conversiones. En este caso, se ignora el pedido de interrupción. Se muestra la figura 5.3 la cual indica el diagrama de flujo del proceso de adquisición.

5.1.3 Diagrama de flujo proceso de adquisición.

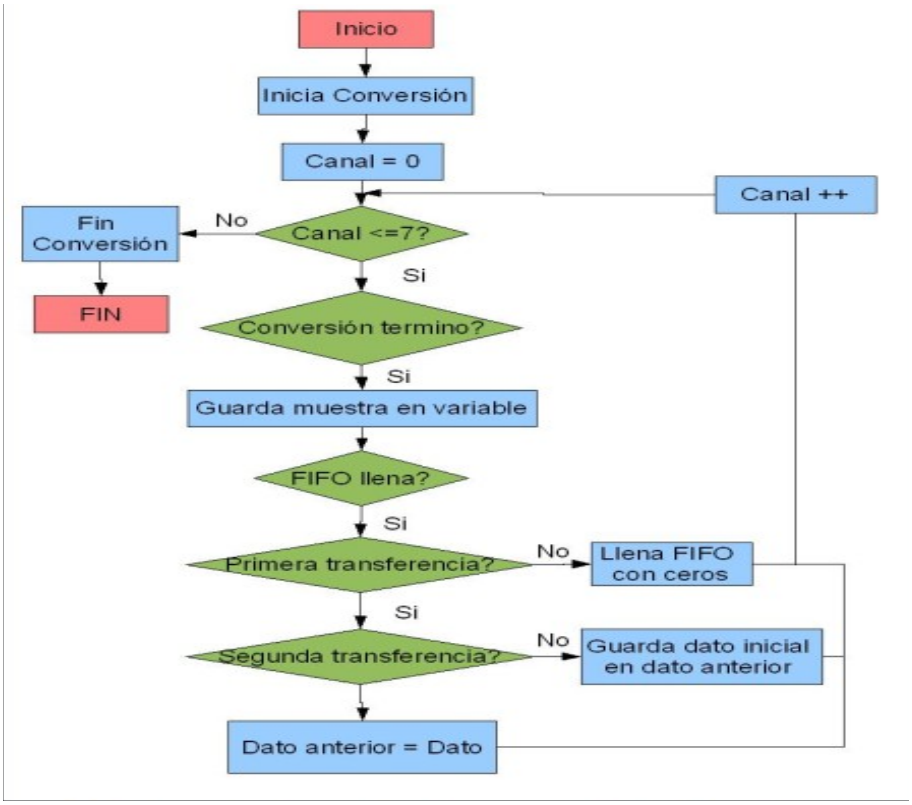

Figura 5.3: Proceso de Adquisición

5.1.4 Proceso de transferencia.

La rutina de transferencia, es ejecutada siempre que FIFO este lleno, simplemente verifica si la última transferencia fue concluida e indica para la comunicación USB se los paquetes de datos están listos, como se muestra en la figura 5.4.

5.1.5 Diagrama de flujo del proceso de transferencia.

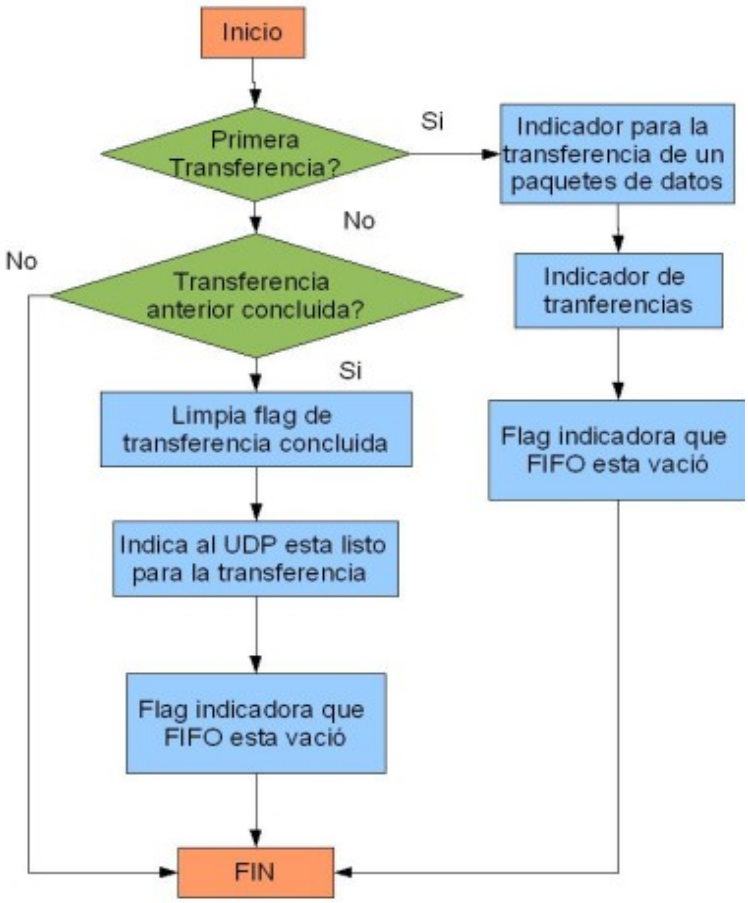

Figura 5.4: Proceso de transferencia.

Una ves que han sido implementados las etapas de adquisición, enumeración USB, transferencia USB y plotaje, el resultado obtenido es el siguiente como se muestra en la figura 5.5.

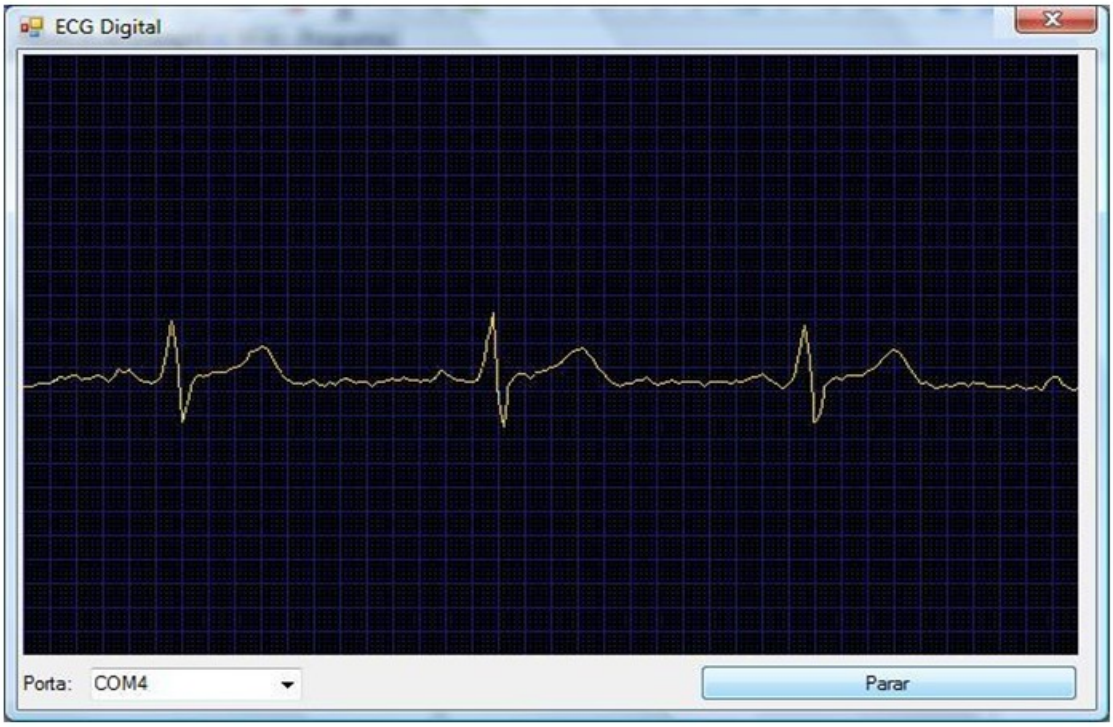

Figura 5.5: Resultados de la integración de los procesos.

En el anexo 1 se muestra el código desarrollado para proceso de enumeración y transmisión para el dispositivo USB.

En el anexo 2 se muestra el código desarrollado para la configuración del conversor análogo digital.

El anexo 3 muestra la configuración de interrupciones y *timer*.

El anexo 4 muestra la configuración de los puestos de entrada y salida.

CAPÍTULO 6:

CONCLUSIONES Y RECOMENDACIONES.

El circuito de amplificación y filtraje de un canal de electrocardiograma (ECG) fue proyectado e implementado para operar aplicando una ganancia de 1000V/V de la señal de entrada, filtrar la señal en una banda de 0.1 Hz a 100 Hz. El resultado del circuito fue satisfactorio considerando que fue implementado su montaje en una placa de matriz de contactos.

El conversor embutido en nuestro micro controlador fue programado para trabajar en modo síncrono, informando el momento de cada digitalización. Lo que prevé que el microcontrolador actuará de manera inmediata a un pedido de interrupción del conversor.

El *firmware* desarrollado para transferir datos para un PC a través de la comunicación USB necesitó la implementación de un proceso de enumeración sencillo, el modelo de comunicación USB utilizado fue para un dispositivo de la clase HID, el papel de cada uno de los registradores utilizados son utilizados para caracterizar al dispositivo HID, transmite la configuración del mismo, al clase, el tipo de buffer a transmitir; el resultado del proceso de implementación fue muy bueno ya que el sistema operativo reconoció al dispositivo enumerado, catalogándolo como listo para su utilización. Quedando como desafió desarrollar los procesos de transmisión de los paquetes de datos.

Se desenvolvió un algoritmo el cual transmitía datos de una manera intuitiva pero se encontró dificultades para solucionar la velocidad de transmisión del dispositivo al PC, ya que se encontró que existe un tiempo elevado de transmisión de cada paquete de datos. Se encontró que cada paquete se transmitía a una velocidad de 64 bytes. Asumiendo que las especificaciones del fabricante para categorizar como un puerto USB 2.0 son correctas, se llego a la conclusión que el algoritmo de comunicación se encuentra errado en su procedimiento. Se paso a estudiar soluciones para conseguir tasas de comunicación mayores, modificando la estructura del algoritmo, reduciendo código y agilitando este proceso; Una solución propuesta fue la de utilizar un sistema simple de modulación delta, en ese caso cada muestra seria transmitida por apenas 8 bits, con perdida mínima en la información.

Para trabajos futuros se puede aprovechar el poder del protocolo USB para crear electrocardiógrafos digitales que adquiera y convierta señales cardiacas en tiempo real en mas de un canal. Esta implementación traerá muchos beneficios ya que los dispositivos USB, son fáciles de utilizar, obtienen energía del mismo puerto de comunicación, se requerirá de un computador personal o un *Laptop* que posea puerto USB, lo cual en el actual momento es una característica fundamental.

Para que el dispositivo ECG sea seguro al paciente, será necesario implementar un circuito de isolación óptica para los electrodos para que estos sean un circuito independiente de los amplificadores, filtros, conversor AD, y micro controlador.

Se recomienda para implementación de futuros trabajos, tomar como requerimientos las normas internacionales y nacionales para equipos ECG, dentro de ellas se especifican características a considerar, como lo son es el número de canales mínimos a utilizar, las características del conversor AD, el número de bytes necesarios para ser considerado un equipo hospitalario, cuáles son las características de seguridad referente a isolación y aterra miento del dispositivo que son requeridas a nivel médico, de esta forma poder categorizar al equipo.

Se recomienda tomar como base a este prototipo ya que en él se han conseguido con éxito llegar a los objetivos planteados, la continuidad que se de estos proyectos por parte de estudiantes y profesores a los cuáles el interés por el desarrollo de tecnologías propias, es uno de las principales virtudes que deben tener las universidades y se debería constituir en una política nacional para el desarrollo del país.

El desarrollo de este prototipo nos da una idea de la complejidad que existe en este tipo de equipos médicos, llegamos a la conclusión que mediante las técnicas adecuadas y solucionando problemas inconclusos en este prototipo se puede lograr desarrollar un equipo médico que cumpla con todas normas requeridas y adicionalmente poseer el nivel requerido de calidad y complejidad para ser apreciado por el área médica.

Se recomienda que al prototipo presentado sea mudado de un conversor análogo digital por otro de mayor resolución, de preferencia por un externo que posea comunicación serial con el microcontrolador. Adicionalmente recomendamos la implementación de un número mayor de canales de adquisición analógicos.

Como conclusión final se ratifica que es perfectamente factible el desarrollo de este tipo de tecnologías, a favor del área médica ya que ayudaran al desarrollo de los pueblos.

ANEXOS

FIRMWARE MICRO PROCESADOR.

Anexo 1: Proceso de enumeración y transmisión para el dispositivo USB.

```
//  ******************************************************
//                Header Files
//  ******************************************************
#include "AT91SAM7S256.h"
#include "board.h"

/*-----------------*/
/* Type definitions*/
/*-----------------*/
#define BYTE            unsigned char
#define WORD            unsigned int

/* Definiciones globales*/
unsigned long     tickcount = 0;                                        //
global variável counts interrupts
#define RC_SIZE 599                                          // RC Timer Size
(10kHz)
#define ADC_RESOLUTION 16                                    // resolução do
ADC (16)
#define USB_CHANNELS 1                                       // 1 A 8 canais
de entrada
#define ADC_CHANNELS 4                                       // 1 A 8 canais
de entrada
#define SAM_FREQ (ADC_CHANNELS * 10000)         // Sampling frequency in Hz
(canais * freq) = 40khz
#define SAM_1MS (SAM_FREQ/1000)                       // # de amostras em
1ms = 40
#define BUFFER_SIZE (SAM_1MS * USB_CHANNELS)    // Buffer (1ms de amostras
por canal) = 40
```

```
#define TIM0_INT_PRIOR 7
#define TC_CLKS 0x7


#define EP0_SIZE 0x08                           // "tamanho do EP0": Define
o limite de bytes para ser enviados ou recebedos o valor maximo é 8
#define EP1_SIZE (0x0002 * BUFFER_SIZE) // amostras de 1 quadro de 1ms * 2
bytes
#define EP2_SIZE 0x08                           // valor maximo 8
#define EP3_SIZE 0x08                           // valor maximo 8


/* General Macros */
#define MIN(a, b) (((a) < (b)) ? (a) : (b))
#define EP_NUMBER 1
#define BUTTON_SAMPLING 1700 // Sampling frequency of buttons

#define BL_OFF(pio) ((pio) & SW1)
#define BR_OFF(pio) ((pio) & SW2)
#define BU_OFF(pio) ((pio) & SW3)
#define BD_OFF(pio) ((pio) & SW4)

#define BL_ON(pio) (!BL_OFF(pio))
#define BR_ON(pio) (!BR_OFF(pio))
#define BU_ON(pio) (!BU_OFF(pio))
#define BD_ON(pio) (!BD_OFF(pio))
#define CLICKL_ON 1
#define CLICKR_ON 2


// Descriptor do dispositivo
unsigned char HID_REPORT[]=
{

      // HID Report //
      0x06,0x00,0xFF,
      //usage page
    0x09,0x01,                                                          //Usa
ge I/O
      0xA1,0x01,
      //Collection
      0x19,0x01,
      //Usage_minimun
      0x29,0x08,
      //Usage_MAximun
    0x15,0x80,                                                          //Log
ical_minimun
    0x25,0x7F,                                                          //Log
ical_maximun
      0x75,0x08,
      //Report_size
```

```
      0x95,0x40,
      //Report_count
      0x81,0x02,
      //Input
      0x19,0x01,
      //Usage_minimun
      0x29,0x08,
      //Usage_maximun
      0x91,0x02,
      //Output
      0xC0
      //End_collection
};
//
const char DeviceDescriptor[]=
{
      0x12,
      //bLength = tamahno do descriptor 18 bytes
      0x01,
      //bDescriptorType (DEVICE) = num descriptor de dispositivo vale sempre
1.
      0x10,0x01,
      //bcdUSB (USB 2.0) = versão USB 0200h o byte menos significativo vem
primero
      0x00,
      //bDeviceClass = que dispositivo é -- colocamos zero
      0x00,
      //bDeviceSubClass -- colocamos zero
      0x00,
      //bDevice -- colocamos zero
      0x08,//
      //bMaxPacketSize0 (Endpoint0 Max Size) = o endpoint0 foi definido em
mapa.h com o valor de 8
      0xFF,0xFF,
      //idVendor = info do fornecedor
      0x00,0x00,
      //idProduct = info do fornecedor
      0x01,0x00,
      //bcdDevice = info do fornecedor
      0x00, //pode ser 0x01
      //iManufacturer String -- info do fornecedor-- colocamos 1 atmel
usb_if
      0x00,
      //iProduct String -- info do fornecedor-- colocamos 2 atmel usb_if
      0x00,
      //iSerialNumber String -- info do fornecedor-- colocamos 3 atmel
usb_if
      0x01,
      //bNumConfigurations = define o número de descriptores de configuração
que e precisa
 };
 //
```

```
const char ConfigurationDescriptor[]=
{
      0x09,
      //bLength = deve ter o valor de 9
      0x02,
      //bDescriptorType (CONFIGURATION) = deve ter o valor de 2
      0X29,0x00,//o//22h so para um endpoint    y 0x29 para
dos//bTotalLength = tamanho total dos descriptores-- colocamos 64 atmel
usb_if
      0x01,
      //bNumInterfaces = são as interfaces para ser analizadas
      0x01,
      //bConfigurationValue = é à qual das número de configurações o
descriptor se refere
      0x00,
      //iConfiguration String = é um string alfanumerica associada à
configuração
      0x80,//O//0xC0/olho com esto podemos trocar//10000000 //bmAttributes
(Bus Powered, no Remote wakeup capability) = define augumas carateristicas,
neste caso colocamos 1000000 que é (precisa de alimentação do cabo)
      0x32,
      //Maximun Power consum(0x32=50=100mA) = é a metade da corrente
elétrica que se precisa

      //descriptor de Interface // // para el caso de utilizar la classe
HID.
      0x09,
      //bLength = deve ter o valor de 9
      0x04,
      //bDescriptorType (INTERFACE) = deve ter o valor de 4
      0x00,
      //bInterfaceNumber = é à qual das número de configurações o descriptor
se refere
      0x00,
      //bAlternateSetting = deve ter o valor de 0
      0x02,//o//0x01 solo un ep//para 0x02 dos ep          //bNumEndpoints
(Uses EP0) = para utilizar dois end points ep0 ep1 ep2
      0x03,
      //bInterfaceClass (HID) = Já que tamos trabalhando com a classe HID
      0x00,//O//0x00-->HID//0x01 boot subclass for mause//bInterfaceSubClass
(HID_CONTROL) = pode ser 0
      0x00,//O//0x00-->HID//0x02 protocol code mause //bInterfaceProtocol =
o protocolo pode ser 0
      0x00,
      //iInterface String = o nome da interface pode ser 0

      //descriptor HID//                                                  //os
valores já tan definidos pela classe HID.
      0x09,
      //bLength = deve ter o valor de 9.
      0x21,
      //bDescriptorType (INTERFACE) = deve ter o valor de 21.
```

```
	0x10,0x01,    // versão do HID 1.1             //bversion = versão
HID
	0x00,
	//bcodec = o codigo do pais.
	0x01,
	//descriptores restantes
	0x22,
	//tamanho do descriptor de relatorio
	sizeof(HID_REPORT),
	//wItemLength
	0x00,

	// IN Endpoint Descriptor //
	0x07,
	//bLength = deve ter o valor de 7
	0x05,
	//bDescriptorType (ENDPOINT) = deve ter o valor de 5
	0x81,//olho com esto no usb_if dice 1
	//bEndpointAddress (saida do device ou 1 IN to host) (10000001b)
	0x03,
	//bmAttributes (03H INTERRUP)
	0x40,0x00,//o//0x40,0x00///
	//wMaxPacketSize (# bytes per packet)
	0x0A,
	//bInterval

	//Out Endpoint Descriptor //
	0x07,
	//bLength
	0x05,
	//bDescriptorType (ENDPOINT) = deve ter o valor de 5
	0x02,
	//bEndointAddress (OUT 2)(00000010b)
	0x03,
	//bmAttributes (03H INTERRUP)
	0x0A,0x00,
	//wMaxPacketSize (# bytes per packet)
	0x01
	//bInterval
};
//

int flagMode = 0;

//adc module
signed short sampleBuffer[BUFFER_SIZE];
char bufferFull = 0;
char adcChannel = 0;
char conversionOver = 1;
unsigned short pIn = 0, pOut = BUFFER_SIZE;
unsigned short contador = 0;
char conversorAtual = 0;
```

```
//////////////////////////
/* Basic USB Parameters */
//////////////////////////

/*Descriptor do dispositivo*/

#define HID_IN 0x01                              // Endpoint de HID é uma das
configurações que o número necessita ooooojoooooo

/* Descriptor types */

// Os tipos de descriptores estão definidos

#define STD_GET_STATUS_ZERO         0x0080
#define STD_GET_STATUS_INTERFACE    0x0081
#define STD_GET_STATUS_ENDPOINT     0x0082

#define STD_CLEAR_FEATURE_ZERO           0x0100
#define STD_CLEAR_FEATURE_INTERFACE 0x0101
#define STD_CLEAR_FEATURE_ENDPOINT  0x0102

#define STD_SET_FEATURE_ZERO        0x0300
#define STD_SET_FEATURE_INTERFACE   0x0301
#define STD_SET_FEATURE_ENDPOINT    0x0302

#define STD_SET_ADDRESS                  0x0500
#define STD_GET_DESCRIPTOR               0x0680
#define STD_SET_DESCRIPTOR               0x0700
#define STD_GET_CONFIGURATION            0x0880
#define STD_SET_CONFIGURATION            0x0900
#define STD_GET_INTERFACE                0x0A81
#define STD_SET_INTERFACE                0x0B01
#define STD_SYNCH_FRAME                  0x0C82

/* HID Class Specific Request Code */
#define CLASS_GET_HID_DESCRIPTOR        0x0681
#define CLASS_SET_IDLE                  0x0A21
#define CLASS_SET_PROTOCOL                     0X0B21
#define CLASS_GET_PROTOCOL                     0X03A1
//  ***************************************************

struct _AT91S_HID HID;///

char firstTransfer = 1;
char packetSent = 1;
//---------------------------------------*/
//                                MAIN
//---------------------------------------*/
void inicio(void);
static void wait ( int tempo );
void blink(void);
```

```
void ADC_init(void);

typedef struct _AT91S_HID
{
    // Private members
    AT91PS_UDP pUdp;
    unsigned char currentConfiguration; //
    unsigned char currentSetting;
    unsigned char protocol;
    // Public Methods:
    unsigned char (*IsConfigured)(struct _AT91S_HID *HID);
      void (*Write)(struct _AT91S_HID *pCdc, char button, char x, char y);
      void (*BufferReady)(struct _AT91S_HID pHid);
      unsigned char(*Read)(struct _AT91S_HID *pCdc, char pakt);

} AT91S_HID, *AT91PS_HID;
//

//USB FUNCTIONS
void AT91F_USB_OPEN(void);
AT91PS_HID AT91F_HID_OPEN(AT91PS_HID pHID,AT91PS_UDP pUdp); //
static unsigned char AT91F_UDP_IsConfigured(AT91PS_HID);//
static void AT91F_HID_Write(AT91PS_HID pHid, char button, char x, char y);//
static void AT91F_USB_BufferReady(AT91PS_HID pHid);
static unsigned char AT91F_HID_Read(AT91PS_HID pHID,char pakt);
static void AT91F_USB_SendData(AT91PS_UDP pUdp, const char *pData, unsigned
int length);
void AT91F_USB_SendZlp(AT91PS_UDP pUdp);
void AT91F_USB_SendStall(AT91PS_UDP pUdp);
static void AT91F_USB_Enumerate(AT91PS_HID);//

//  ******************************************************
//               Function Prototypes
//  ******************************************************
void Timer0IrqHandler(void);
void FiqHandler(void);
void Tim0Init(void);
//  ******************************************************
//               External References
//  ******************************************************
extern      void LowLevelInit(void);
extern      void TimerSetup(void);
extern      unsigned enableIRQ(void);
extern      unsigned enableFIQ(void);

//  ******************************************************
//              Global varíables
//  ******************************************************
unsigned int      FiqCount = 0;           // global uninitialized varíable

int                     q;                          // global
uninitialized varíable
```

```
int                     r;                              // global
uninitialized varíable
int                     s;                              // global
uninitialized varíable
int                     m = 2;                          // global initialized
varíable
int                     n = 3;                          // global initialized
varíable
int                     o = 6;                          // global initialized
varíable

struct comms {
      int         nBytes;
      char  *pBuf;
      char  Buffer[32];
}  Channel = {5, &Channel.Buffer[0], {"Faster than a speeding bullet"}};


//  ******************************************************
//                      MAIN
//  ******************************************************/
int   main (void)
{
      volatile AT91PS_PIO           pPIO = AT91C_BASE_PIOA;

      //unsigned char reciv;
      unsigned int pioStatus;
      *AT91C_RTTC_RTMR = BUTTON_SAMPLING;

      ///////////////////////
      // Initialize the Atmel AT91SAM7S256 (watchdog, PLL clock, default
interrupts, etc.)
      LowLevelInit();
      ///////////////////////

      inicio();

      AT91F_USB_OPEN();

      while (!HID.IsConfigured(&HID));
      pPIO->PIO_CODR=(LED2);

      ADC_init();

      Tim0Init();
      blink();

      int i=0;
      for(i=0;i<BUFFER_SIZE;i++)
            sampleBuffer[i]=i%ADC_CHANNELS;
      bufferFull=1;
```

```
	int num=0;

	// endless blink loop
	while [1]
	{


			if (HID.IsConfigured(&HID) && ((*AT91C_RTTC_RTSR) &
AT91C_RTTC_RTTINC))
			{
				pioStatus = *AT91C_PIOA_PDSR;
				if (BL_ON(pioStatus))
				{
					HID.Write(&HID,num,num,num);
					blink();
				}
			}
			else if (!(HID.IsConfigured(&HID)))
				{
					pPIO->PIO_SODR=(LED1);
				}

	}
}

void AT91F_USB_OPEN(void)//
{
	// Configura o divisor do PLL para o barremento USB.
	AT91C_BASE_CKGR->CKGR_PLLR|=AT91C_CKGR_USBDIV_1; // (CKGR) Divider
output is PLL clock output divided by 2
	//CONFIGURA O CLOCK DO usb DEVICE (UDP)
	AT91C_BASE_PMC->PMC_SCER=AT91C_PMC_UDP; //HABILITAÇÃO DO CLOCK
	//CONFIGURA O PERIPHERL CLOCK CONTROLLER
	AT91C_BASE_PMC->PMC_PCER=(1<<AT91C_ID_UDP); //HABILITAÇÃO DO PERIPHERL
CLOCK CONTROLLER PARA USB
	//HABILITAÇÃO UDP PULL-UP// MEDIANTE CONFIGURAÇÃO DO PA16
	// Enable UDP PullUp (USB_DP_PUP) : enable & Clear of the
corresponding PIO
    // Set in PIO mode and Configure in Output
	AT91C_BASE_PIOA->PIO_PER=(AT91C_PIO_PA25);
	AT91C_BASE_PIOA->PIO_OER=(AT91C_PIO_PA25);
	// Clear for set the Pul up resistor
	// COLOCA EM Alto AS SALIDAS //jeito completo
	AT91C_BASE_PIOA->PIO_SODR=(AT91C_PIO_PA25);
	//INICIA A ESTRUTURA HID
	AT91F_HID_OPEN(&HID,AT91C_BASE_UDP);
}
AT91PS_HID AT91F_HID_OPEN(AT91PS_HID pHID, AT91PS_UDP pUdp)//
{

	pHID->pUdp = pUdp;//
	pHID->currentConfiguration = 0;//
```

```
	pHID->currentSetting = 0;//
	pHID->protocol = 1;//
	pHID->IsConfigured = AT91F_UDP_IsConfigured;//
	pHID->Write = AT91F_HID_Write;//


	return pHID;

}
static unsigned char AT91F_UDP_IsConfigured(AT91PS_HID pHID)//
{
	AT91PS_UDP pUDP = pHID->pUdp;
	AT91_REG isr = pUDP->UDP_ISR; //UDP_ISR INTERUP STATUS REGISTER // isr
almacenador

	if (isr & AT91C_UDP_ENDBUSRES) //(UDP-ENDBUSRES) USB End Of Bus Reset
Interrupt
	{
		pUDP->UDP_ICR = AT91C_UDP_ENDBUSRES;//(ICR) Interrupt Clear
Register

		// reinicia todos os endpoints
		pUDP->UDP_RSTEP = (unsigned int) -1;
		pUDP->UDP_RSTEP = 0;

		// Habilita a função
		pUDP->UDP_FADDR = AT91C_UDP_FEN;

		// Configura endpoint 0
		pUDP->UDP_CSR[0] = (AT91C_UDP_EPEDS | AT91C_UDP_EPTYPE_CTRL);
	}
	else if (isr & AT91C_UDP_EPINT0)
	{
		AT91F_USB_Enumerate(pHID);
	}

	return pHID->currentConfiguration;
}
static void AT91F_USB_Enumerate(AT91PS_HID pHID)//
{


	AT91PS_UDP pUDP = pHID->pUdp;

	unsigned char bmRequestType, bRequest;
	unsigned short wValue, wIndex, wLength, wStatus;

	if ( !(pUDP->UDP_CSR[0] & AT91C_UDP_RXSETUP) )// UDP_CSR (UDP)
Endpoint Control and Status Register
	{
		return;
	}// AT91C_UDP_RXSETUP (UDP) Sends STALL to the Host (Control
```

```
endpoints)


      bmRequestType     = pUDP->UDP_FDR[0];           // Endpoint0 FIFO Data
Register
      bRequest          = pUDP->UDP_FDR[0];                 // Endpoint0
FIFO Data Register
      wValue                  = (pUDP->UDP_FDR[0] & 0xFF);
      wValue               |= (pUDP->UDP_FDR[0] << 8);      //wvalue
      wIndex                  = (pUDP->UDP_FDR[0] & 0xFF);
      wIndex               |= (pUDP->UDP_FDR[0] << 8);      //windex
      wLength           = (pUDP->UDP_FDR[0] & 0xFF);
      wLength        |= (pUDP->UDP_FDR[0] << 8);      //wlength

      if (bmRequestType & 0x80)
      {
            pUDP->UDP_CSR[0] |= AT91C_UDP_DIR;                    // udp_csr
Endpoint0 Control and Status Register
            while ( !(pUDP->UDP_CSR[0] & AT91C_UDP_DIR) );// (UDP) Transfer
Direction
      }

      pUDP->UDP_CSR[0] &= ~AT91C_UDP_RXSETUP;
      while ( (pUDP->UDP_CSR[0] & AT91C_UDP_RXSETUP) );

      // Trata os pedidos padrões da specification USB Rev 2.0
      switch ((bRequest << 8) | bmRequestType)
      {

            case STD_GET_DESCRIPTOR:

                  if (wValue == 0x100) // descriptor de dispositivo
                  {
                        AT91F_USB_SendData(pUDP, DeviceDescriptor,
MIN(sizeof(DeviceDescriptor), wLength));
                  }
                  else if (wValue == 0x200) // descriptor de configuração
                        {


AT91F_USB_SendData(pUDP,ConfigurationDescriptor,MIN(sizeof(ConfigurationDesc
riptor),wLength));
                        }
                        else
                              {
                                    AT91F_USB_SendStall(pUDP);
                              }
            break;

            case STD_SET_ADDRESS://
                  AT91F_USB_SendZlp(pUDP);
```

```
                  pUDP->UDP_FADDR = (AT91C_UDP_FEN | wValue);
                  pUDP->UDP_GLBSTATE = (wValue) ? AT91C_UDP_FADDEN : 0;

            break;

            case STD_SET_CONFIGURATION://
                  pHID->currentConfiguration = wValue;
                  AT91F_USB_SendZlp(pUDP);
                  pUDP->UDP_GLBSTATE = (wValue) ? AT91C_UDP_CONFG :
AT91C_UDP_FADDEN;
                  pUDP->UDP_CSR[EP_NUMBER] = (wValue) ? (AT91C_UDP_EPEDS |
AT91C_UDP_EPTYPE_BULK_IN) : 0;
            break;

            case CLASS_GET_HID_DESCRIPTOR://
                  //if(wIndex == 0X2200) // return device descriptor 0X2200
                  //{

                        AT91F_USB_SendData(pUDP,(const char *)
HID_REPORT,MIN((sizeof(HID_REPORT)),wLength));
                  //}
                  //else if (wValue == 0x2100)//return HID descriptor 0X2100
                        //{


AT91F_USB_SendData(pUDP,&ConfigurationDescriptor[18],MIN(0x09,wLength));
                        //}
                  //else
                        //{
                              AT91F_USB_SendStall(pUDP);
                        //}
            break;

            case STD_GET_CONFIGURATION://??
                  AT91F_USB_SendData(pUDP,(char *) &(pHID-
>currentConfiguration),sizeof(pHID->currentConfiguration));
            break;

            case CLASS_SET_PROTOCOL://??
                  pHID->protocol = wValue;
                  AT91F_USB_SendZlp(pUDP);
            break;

            case CLASS_GET_PROTOCOL://??
                  AT91F_USB_SendData(pUDP, (char *) &(pHID->protocol),
sizeof(pHID->protocol));
            break;

            case STD_GET_STATUS_ZERO://??
                  wStatus = 0;
                  AT91F_USB_SendData(pUDP, (char *) &wStatus,
sizeof(wStatus));
```

```
            break;

            case STD_GET_STATUS_INTERFACE://??
                  wStatus = 0;
                  AT91F_USB_SendData(pUDP, (char *) &wStatus,
sizeof(wStatus));
            break;

            case STD_GET_STATUS_ENDPOINT://??
                  wStatus = 0;
                  wIndex &= 0x0F;
                  if ((pUDP->UDP_GLBSTATE & AT91C_UDP_CONFG) && (wIndex <=
3))
                  {
                        wStatus = (pUDP->UDP_CSR[wIndex] &
AT91C_UDP_EPEDS) ? 0 : 1;
                        AT91F_USB_SendData(pUDP, (char *) &wStatus,
sizeof(wStatus));
                  }
                  else if ((pUDP->UDP_GLBSTATE & AT91C_UDP_FADDEN) &&
(wIndex == 0))
                  {
                        wStatus = (pUDP->UDP_CSR[wIndex] &
AT91C_UDP_EPEDS) ? 0 : 1;
                        AT91F_USB_SendData(pUDP, (char *) &wStatus,
sizeof(wStatus));
                  }
                  else
                        AT91F_USB_SendStall(pUDP);
            break;

            case STD_SET_FEATURE_ZERO://??
                  AT91F_USB_SendStall(pUDP);
            break;

            case STD_SET_FEATURE_INTERFACE://??
                  AT91F_USB_SendZlp(pUDP);
            break;

            case STD_SET_FEATURE_ENDPOINT:
                  wIndex &= 0x0F;
                  if ((wValue == 0) && wIndex && (wIndex <= 3))
                  {
                        pUDP->UDP_CSR[wIndex] = 0;
                        AT91F_USB_SendZlp(pUDP);
                  }
                  else
                        AT91F_USB_SendStall(pUDP);
            break;

            case STD_CLEAR_FEATURE_ZERO:
                  AT91F_USB_SendStall(pUDP);
```

```
            break;

            case STD_CLEAR_FEATURE_INTERFACE:
                  AT91F_USB_SendZlp(pUDP);
            break;

            case STD_CLEAR_FEATURE_ENDPOINT:
                  wIndex &= 0x0F;
                  if ((wValue == 0) && wIndex && (wIndex <= 3))
                  {
                  if (wIndex == 1)
                        pUDP->UDP_CSR[1] = (AT91C_UDP_EPEDS |
AT91C_UDP_EPTYPE_BULK_OUT); //UDP_CSR // Endpoint Control and Status
Register
                  else if (wIndex == 2)
                        pUDP->UDP_CSR[2] = (AT91C_UDP_EPEDS |
AT91C_UDP_EPTYPE_BULK_IN);
                  else if (wIndex == 3)
                        pUDP->UDP_CSR[3] = (AT91C_UDP_EPEDS |
AT91C_UDP_EPTYPE_ISO_IN);
                  AT91F_USB_SendZlp(pUDP);
                  }
                  else
                        AT91F_USB_SendStall(pUDP);
            break;

            case CLASS_SET_IDLE://
                  AT91F_USB_SendZlp(pUDP);
            break;

      default:
            AT91F_USB_SendStall(pUDP);
          break;
      }

}
static void AT91F_USB_SendData(AT91PS_UDP pUdp, const char *pData, unsigned
int length)//
{
      unsigned int cpt = 0;
      AT91_REG csr;
      do
      {
            cpt = MIN(length, 8);
            length -= cpt;

            while (cpt--)
            pUdp->UDP_FDR[0] = *pData++;

            if (pUdp->UDP_CSR[0] & AT91C_UDP_TXCOMP)
            {
                  pUdp->UDP_CSR[0] &= ~(AT91C_UDP_TXCOMP);
```

```
                while (pUdp->UDP_CSR[0] & AT91C_UDP_TXCOMP);
            }
            pUdp->UDP_CSR[0] |= AT91C_UDP_TXPKTRDY;

            do
            {
                csr = pUdp->UDP_CSR[0];

                // estagio Data IN interrompido por um Status OUT
                if (csr & AT91C_UDP_RX_DATA_BK0)
                {
                    pUdp->UDP_CSR[0] &= ~(AT91C_UDP_RX_DATA_BK0);
                    return;
                }
            } while ( !(csr & AT91C_UDP_TXCOMP) );

        } while (length);

        if (pUdp->UDP_CSR[0] & AT91C_UDP_TXCOMP)
        {
            pUdp->UDP_CSR[0] &= ~(AT91C_UDP_TXCOMP);
            while (pUdp->UDP_CSR[0] & AT91C_UDP_TXCOMP);
        }
}
void AT91F_USB_SendStall(AT91PS_UDP pUdp)//
{
        pUdp->UDP_CSR[0] |= AT91C_UDP_FORCESTALL;

        //while ( !(pUdp->UDP_CSR[0] & AT91C_UDP_ISOERROR) );

        pUdp->UDP_CSR[0] &= ~(AT91C_UDP_FORCESTALL | AT91C_UDP_ISOERROR);

        while (pUdp->UDP_CSR[0] & (AT91C_UDP_FORCESTALL |
AT91C_UDP_ISOERROR));
}
void AT91F_USB_SendZlp(AT91PS_UDP pUdp)//
{
        pUdp->UDP_CSR[0] |= AT91C_UDP_TXPKTRDY;

        while ( !(pUdp->UDP_CSR[0] & AT91C_UDP_TXCOMP) );

        pUdp->UDP_CSR[0] &= ~(AT91C_UDP_TXCOMP);

        while (pUdp->UDP_CSR[0] & AT91C_UDP_TXCOMP);
}
static void AT91F_HID_Write(AT91PS_HID pHid, char button, char x, char y)
{
        AT91PS_UDP pUdp = pHid->pUdp;

        // Send report to the host
        //pUdp->UDP_FDR[EP_NUMBER] = button;
        //pUdp->UDP_FDR[EP_NUMBER] = x;
```

```
	//pUdp->UDP_FDR[EP_NUMBER] = y;
	int a;
	for(a=0; a<62; a++)
	{
		pUdp->UDP_FDR[EP_NUMBER]=a+27;
	}
	pUdp->UDP_FDR[EP_NUMBER]=0xFF;
	pUdp->UDP_FDR[EP_NUMBER]=0xFF;


	pUdp->UDP_CSR[EP_NUMBER] |= AT91C_UDP_TXPKTRDY;

	// Wait for the end of transmission
	while ( !(pUdp->UDP_CSR[EP_NUMBER] & AT91C_UDP_TXCOMP) )
		AT91F_UDP_IsConfigured(pHid);
	// Clear AT91C_UDP_TXCOMP flag
	if (pUdp->UDP_CSR[EP_NUMBER] & AT91C_UDP_TXCOMP)
	{
		pUdp->UDP_CSR[EP_NUMBER] &= ~(AT91C_UDP_TXCOMP);
		while (pUdp->UDP_CSR[EP_NUMBER] & AT91C_UDP_TXCOMP);
	}
}

static void AT91F_USB_BufferReady(AT91PS_HID pHid)
{
	AT91PS_UDP pUdp = pHid->pUdp;
	if (pUdp->UDP_CSR[EP_NUMBER] & AT91C_UDP_TXCOMP)
	{
		pUdp->UDP_CSR[EP_NUMBER] &= ~(AT91C_UDP_TXCOMP);
		while (pUdp->UDP_CSR[EP_NUMBER] & AT91C_UDP_TXCOMP);
		packetSent = 1;
		pUdp->UDP_CSR[EP_NUMBER]|=AT91C_UDP_TXCOMP;
	}
}

static unsigned char AT91F_HID_Read(AT91PS_HID pHID, char pakt)
{
	AT91PS_UDP pUdp = pHID->pUdp;
	char reciv;
	//char long data;
	if (pUdp->UDP_CSR[2]&AT91C_UDP_RX_DATA_BK0)
	{
			reciv=pUdp->UDP_FDR[2];
			while(!(pUdp->UDP_CSR[2] & AT91C_UDP_RX_DATA_BK0))
			AT91F_UDP_IsConfigured(pHID);
			pUdp->UDP_CSR[2] &=~(AT91C_UDP_RX_DATA_BK0);
	}
	//data=AT91C_UDP_RXBYTECNT;
	return reciv;
}
```

Anexo 2: Configuración del conversor análogo digital.

```
void ADC_init(void)
{
      volatile AT91PS_ADC           pADC = AT91C_BASE_ADC;              //
pointer to ADC register structure
      volatile AT91PS_PIO           pPIO = AT91C_BASE_PIOA;             //
pointer to PIO register structure


      pADC->ADC_MR=(    (AT91C_ADC_TRGEN_DIS)|
      //DESABILITA TRIGGER DE HARDWARE
                        (AT91C_ADC_TRGSEL_TIOA0)|
      //SELEÇÃO DE TRIGGER
                        (AT91C_ADC_LOWRES_10_BIT)|
      //RESOLUÇÃO DE 10 BITS
                        (AT91C_ADC_SLEEP_NORMAL_MODE)|
      //MODO NORMAL
                        (AT91C_ADC_PRESCAL & (5<<8))|       //ADC
PRESCALER = 5 (ADCCLOCK = 4 MHz)
                        (AT91C_ADC_STARTUP&(3<<16))|        //TEMPO DE
INICIALIÇÃO DO ADC = 8 (18 us)
                        (AT91C_ADC_SHTIM&(8<<24)));
      //TEMPO DE INICIALIÇÃO DO SH = 3 (1 us)

      pADC->ADC_CHER=0xF0;

      pPIO->PIO_ASR = 0;
      pPIO->PIO_BSR = 0;
      pPIO->PIO_PDR = (0 | 0); // Set in Periph mode

}

void ADC_convert(void)
{
      volatile AT91PS_ADC           pADC = AT91C_BASE_ADC;              //
pointer to ADC register structure
      volatile AT91PS_PIO           pPIO = AT91C_BASE_PIOA;             //
pointer to PIO register structure

      pADC->ADC_CR=AT91C_ADC_START;
      flagMode=0;

      for(adcChannel=0; adcChannel<ADC_CHANNELS; adcChannel++)
            {
                  if(firstTransfer)
                        {
                              sampleBuffer[pIn]=pIn % ADC_CHANNELS;
                              blink();
                        }
                        else
```

```
                                {
                                      sampleBuffer[pIn]=(signed short)(((pADC-
>ADC_LCDR & AT91C_ADC_DATA)<<6)-0x8000);
                                }
                  if(++pIn == BUFFER_SIZE)
                  {
                        bufferFull = 1;
                        pIn = 0;
                  }
            }

            conversionOver = 1;
}
```

Anexo 3: Configuración de interrupciones y timer.

```
void Tim0Init(void)
{
      volatile AT91PS_PMC           pPMC = AT91C_BASE_PMC;
      volatile AT91PS_TC            pTC0 = AT91C_BASE_TC0;
      volatile AT91PS_AIC           pAIC = AT91C_BASE_AIC;

      unsigned int dummy = 0;

      pPMC->PMC_PCER=1<<AT91C_ID_TC0; //HABILITA CLOCK DO TIMER0

      pTC0->TC_CCR=AT91C_TC_CLKDIS;   //DESABILITA EL CLOCK
      pTC0->TC_IDR=0xFFFFFFFF;            //DESABILITA LAS INTERUPCIONES

      dummy=pTC0->TC_SR;                        //LIMPIA LOS REGISTROS DE
ESTATUS TIMER
      dummy=dummy;                              //solo para eliminar un
warnig.

      pTC0->TC_RC=RC_SIZE;          //selecciona do tamanho delcontador int.

      pTC0->TC_CMR=AT91C_TC_CLKS_TIMER_DIV2_CLOCK|AT91C_TC_CPCTRG|
AT91C_TC_WAVE; //SELECCIONA MODO DE CUENTA.

      pTC0->TC_CCR=AT91C_TC_CLKEN|AT91C_TC_SWTRG; //HABILITA TIMER0

      //----------------------CONFIGURA E HABILITA LA
INTERUP-----------------//

      pAIC->AIC_IDCR=1<<AT91C_ID_TC0;  //DESABILITA A INTRRUP DO CONTROLADOR
DE INTERUU

      pAIC->AIC_SVR[AT91C_ID_TC0]=(int)Timer0IrqHandler; //setea la
direccion en el vector de interrup
```

```
      //pAIC->AIC_SMR[AT91C_ID_TC0]=AT91C_AIC_SRCTYPE_INT_EDGE_TRIGGERED|
TIM0_INT_PRIOR; //CONFIGURA O MODO (SENSIBILIDADE Y PRIORIDADE)
      pAIC->AIC_SMR[AT91C_ID_TC0]=(0x1<<5)|(TIM0_INT_PRIOR); //CONFIGURA O
MODO (SENSIBILIDADE Y PRIORIDADE)

      pAIC->AIC_ICCR=1<<AT91C_ID_TC0;   //LIMPIA EL FLAG DE INTERRUP

      pAIC->AIC_IECR=1<<AT91C_ID_TC0;          //HABILITA INTERRUP NO TIMER
0 NO AIC

      pTC0->TC_IER=AT91C_TC_CPCS;              //INTERUPCION DE TRIGER
INTERNO (RC) HABILITADA
}


void Timer0IrqHandler (void) {

      volatile AT91PS_TC           pTC = AT91C_BASE_TC0;         // pointer
to timer channel 0 register structure
      volatile AT91PS_PIO          pPIO = AT91C_BASE_PIOA;       // pointer
to PIO register structure
      unsigned int                 dummy;
      // temporary

      dummy = pTC->TC_SR;
      // read TC0 Status Register to clear it
      tickcount++;
      // increment the tick count
      blink();

      if(conversionOver&&!(bufferFull))
      {
            conversionOver=0;
            ADC_convert();
            blink();
      }
      else
            {
                  pPIO->PIO_SODR=LED2;
            }

      if  ((pPIO->PIO_ODSR & LED1) == LED1)
            pPIO->PIO_CODR = LED1;                                    //
turn LED2 (DS2) on
      else
            pPIO->PIO_SODR = LED1;                                    //
turn LED2 (DS2) off
}

void wait( int tempo )
{//* Begin
```

```
	volatile unsigned int waiting_time ;
	for(waiting_time = 0; waiting_time < tempo; waiting_time++){};
}
```

Anexo 4: Inicialización de puertos de entrada y salida.

```
#include "AT91SAM7S256.h"
#include "board.h"

void inicio(void)
{
	volatile AT91PS_PMC		pPMC = AT91C_BASE_PMC;  // pointer to
PMC data structure
	volatile AT91PS_PIO		pPIO = AT91C_BASE_PIOA;	// pointer
to PIO data structure

	// HABILITA-SE CLOCK PORTA PARALELA DE I/O QUE NÓS REQUEREMOS PARA COM
usb
	pPMC->PMC_PCER = (1<<AT91C_ID_PIOA);		//HABILITAÇÃO DO
PERIPHERL CLOCK CONTROLLER PARA PIOA

	pPMC->PMC_PCER = (1<<AT91C_ID_TC0);		// enable Timer0
peripheral clock

	pPIO->PIO_PER = LED_MASK | SW1_MASK;		// PIO Enable
Register - allow PIO to control pins P0 - P3 and pin 19
	pPIO->PIO_OER = LED_MASK;				// PIO
Output Enable Register - sets pins P0 - P3 to outputs
	pPIO->PIO_SODR = LED_MASK;				// PIO Set
Output Data Register - turns off the four LEDs

	// Select PA19 (pushbutton) to be FIQ function (Peripheral B)
	pPIO->PIO_BSR = SW1_MASK;

	pPIO->PIO_ODR=SW1_MASK;
	pPIO->PIO_PDR=SW1_MASK;
}
```

SOFTWARE DESARROLLADO PARA PC.

Anexo 5: Código desarrollado para Visual Basic.

```
Private Const TAMOSTRAGEM = 1000 * (1 / 400)
Private Const TPLOTAGEM = 1000 * (1 / 25)

Private Const TEMPOGRAFICO = 2000 'tiempo em milisegundos
Private Const TENSAOGRAFICO = 5000 'tension em milivolts

Private Const TAMANHOBUFFER = 20480

Private Const FATORCONVERSAO = 5 / 1024

Dim TimerPlot As New System.Timers.Timer() 'define timer mas
preciso

Private EscalaX As Single
Private EscalaY As Single
Private PortaSerial As New SerialPort

Private Buffer(0 To TAMANHOBUFFER - 1) As PointF
Private IndicePlot As Integer = 1
Private IndiceAD As Integer = 0

Private ImagemFundo As Bitmap
Private Temporizador As New System.Diagnostics.Stopwatch()
Private WithEvents BackWorker As New
System.ComponentModel.BackgroundWorker()

Private Sub Form_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    Control.CheckForIllegalCrossThreadCalls = False
     cboPorta_Preencher()
    DrawGrid(Me.PicGraph)
    CreateTimer()
    BackWorker.WorkerSupportsCancellation = True
End Sub

Private Sub cboPorta_Preencher()
```

```
    For Each sp As String In My.Computer.Ports.SerialPortNames
        cboPorta.Items.Add(sp)
    Next
    If cboPorta.Items.Count > 1 Then
        cboPorta.SelectedIndex = 1
    End If
End Sub

Private Sub DrawGrid(ByVal PicBox As PictureBox)
    Dim Bmp As New Bitmap(PicBox.Width, PicBox.Height)
    Dim X As Integer
    Dim Y As Integer

    Dim PassoT As Long
    Dim PassoV As Long
    PicBox.Image = bmp

    Bmp = New Bitmap(PicBox.Width, PicBox.Height)

    PassoT = (PicBox.Width * 10) \ TEMPOGRAFICO
    EscalaX = PassoT
    PassoV = (PicBox.Height * 40) \ TENSAOGRAFICO
    EscalaY = PassoV

    For x = 0 To PicBox.Width - 1 Step passot
        For y = 0 To PicBox.Height - 1 Step passov
            bmp.SetPixel(x, y, Color.MidnightBlue)
        Next
    Next
    For X = 0 To PicBox.Width - 1
        For Y = 0 To PicBox.Height - 1
            If ((Y Mod (5 * PassoV)) = 0) Or ((X Mod (5 *
PassoT)) = 0) Then
                Bmp.SetPixel(X, Y, Color.MidnightBlue)
            End If
        Next
    Next

    PicBox.BackgroundImage = Bmp
    ImagemFundo = Bmp

    EscalaX = TAMOSTRAGEM * PicBox.Width / TEMPOGRAFICO
    EscalaY = PicBox.Height / TENSAOGRAFICO
```

```
End Sub


Private Sub CreateTimer()
    TimerPlot.Interval = TPLOTAGEM
    AddHandler TimerPlot.Elapsed, New
System.Timers.ElapsedEventHandler(AddressOf
Me.TimerPlot_Elapsed)
End Sub
```

BIBLIOGRAFÍA

[1] ZELENOVSKY, Ricardo, MENDONÇA, Alexandre, ***HARDWARE E INTERFACEAMENTO,*** Tercera edición, MZ editora, Rio de Janeiro, abril de 2002, 1031 páginas.

[2] HYDE, John, ***USB DESIGN BY EXAMPLE***, Segunda edición, Intel University Press, Estados Unidos de America, 2005.

[3] http://www.usb.org/developers/devclass_docs, Documentacion de clases.

[4] ATMEL. AT91SAM7S Series Preliminary. [S.l.], Novembro 2006. Disponível en http://www.atmel.com/products.

[5] AXELSON, J, ***USB COMPLETE: EVERYTHING YOU NEED TO DEVELOP USB PERIPHERALS***, tercera edición, Likeview Reasearch LLC, 2005.

[6] LIU, Chunling, WANG, Xu, ***DEVELOPMENT OF THE SYSTEM TO DETECT AND PROCESS ELECTROMYOGRAM SIGNALS,*** Shanghai, China, Septiembre 2005, 4 páginas.

[7] NEUMAN, Michael, ***BIOPOTENTIAL AMPLIFIERS*****,** Biomedical Engineering at Michigan Tech 1996, 16 páginas.

[8] PAN, Jiapu, TOMPKINS, Willis, ***A REAL-TIME QRS DETECTION ALGORITHM,*** IEEE Transactions on Biomedical Enguneering, Marzo 1985, 7 páginas.

[9] BLANCO-VELASCO, M. ,CRUZ-ROLDAN, F. , GODINO-LLORENTE, J.I. , BARNER, K.E., **ECG COMPRESSION WITH RETRIEVED QUALITY GUARANTEED,** 2 páginas.

[10]JALALEDDINE, Sateh, HUTCHENS, Chriswell, STRATTAN, Robert, COBERLY, William, ***ECG DATA COMPRESSION TECHNIQUES-A UNIFIED APPROACH,*** IEEE Transactions on Biomedical Enguneering, abril 1990, 15 páginas.

[11]CAMPELO, Mauricio, MENDES DE AZEVEDO, Fernando, ***ALGORITMO PARA DETECÇÃO PRECISA DA CONTRAÇÃO VENTRICULAR CARDÍACA VISANDO DETERMINAÇÃO DA VRC INTRAOPERATÓRIA,*** IV congreso brasilero de computación-CBComp 2004, 6 páginas.

[12]ARENY, Pallás Ramón, ***ADQUISICIÓN Y DISTRIBUCIÓN DE SEÑALES,*** Marcombo Boixareu Editores Barcelona, 422 páginas.

[13]http://www.olimex.com/dev/index.html