UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE
TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

“Desarrollo e implementación electrónica de un EMG para monitorización de la actividad deportiva”

Titulación: Grado en Ingeniería en Tecnología de la Telecomunicación
Mención: Sistemas Electrónicos
Autor: Dailos Fernando Ramírez Guski
Tutor: Dr. D. Alfonso Medina Escuela

Fecha: Diciembre de 2015

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

“Desarrollo e implementación electrónica de un EMG para monitorización de la actividad deportiva”

HOJA DE FIRMAS

Tutor	Alumno
Fdo.: Dr. D. Alfonso Medina Escuela	**Fdo.: Dailos F. Ramírez Guski**

Fecha: Diciembre de 2015

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

TRABAJO FIN DE GRADO

“Desarrollo e implementación electrónica de un EMG para monitorización de la actividad deportiva”

HOJA DE EVALUACIÓN

Calificación: _____

Presidente

Fdo.:

Vocal

Fdo.:

Secretario/a

Fdo.:

Fecha: Junio 2015

Agradecimientos

En primer lugar quisiera agradecer a D. Alfonso Medina Escuela la oportunidad que me ha brindado para realizar este proyecto y aprender de él, y a la División COM del Instituto Universitario de Microelectrónica Aplicada y sus laboratorios por prestarme las herramientas esenciales para la realización del Trabajo Final de Grado.

A mis padres, porque son mi inspiración y siempre me han regalado su tiempo para darme consejos y ánimos. Y por supuesto el resto de mi familia, ojalá pudieras leer esto abuelo.

A todos y cada uno de mis profesores, desde el colegio hasta la Universidad, ya que de todos he aprendido cosas valiosas.

A todos mis amigos de siempre, Samir, Sergio, Luke, Jean Paul, Nelson y Pamela. Son los mejores amigos que uno puede desear, muchas gracias por ser como son.

A todas las amistades que he establecido en mi vida universitaria, porque me han ayudado a despejar la cabeza y han sido capaces de sacarme sonrisas hasta en los peores momentos, en especial a Jiménez, Horacio, Alex, Sebas y alguno que se me escape.

A la Residencia Universitaria de Tafira y cada unos de sus trabajadores por esos cuatro maravillosos años de alojamiento.

GRACIAS, de corazón.

Tabla de contenido

Lista de Figuras

Lista de Tablas

Introducción

I. Antecedentes

Desde hace décadas se han estado implementando nuevos mecanismos de control y planificación sobre los deportistas ya que el avance de la tecnología en este ámbito está enfocado a mejorar el rendimiento del usuario.

Las nuevas tecnologías se aplican en deportes que necesitan gran precisión de registros, por ejemplo, en las carreras de velocidad [1] en ciclismo y atletismo se utilizan detectores ópticos para registrar tiempos empleados. Estas técnicas no se utilizan solamente en competición, sino también en entrenamientos, pudiendo parametrizar datos fisiológicos, bioquímicos, biomecánicos, etc. permitiendo estimar la evolución del deportista.

El tenis, el golf, distintas disciplinas atléticas, el ciclismo, la natación o la Fórmula 1 (casi una hibridación entre la ingeniería y el deporte) entre otros muchos, llevan varios años mejorando el nivel de los deportistas gracias a la capacidad de los ordenadores de analizar todas las variables que repercuten en el rendimiento, permitiendo así modificar hábitos o indicar ejercicios específicos.

El Real Madrid, gracias al acuerdo firmado con Microsoft, es uno de los primeros clubes que ha dado el paso de incorporar los más modernos sistemas de análisis para monitorizar el estado de la primera plantilla. A través de distintos sensores que los jugadores llevan durante los entrenamientos, se recogen datos de sus movimientos, velocidad, esfuerzo realizado, fatiga acumulada, etcétera. Estos datos son analizados y puestos al servicio del cuerpo técnico para que puedan decidir qué jugadores están en el momento óptimo de forma [2].

Es por ello que la introducción de sensores de tipo "*wearable*" es un concepto que ha revolucionado el mundo del deporte. Estos dispositivos son sistemas electrónicos que pueden ser adheridos o sujetados en diferentes ropas de entrenamiento. Aunque estos sensores han sido introducidos hace más de una década, actualmente entregan una información valiosísima a la persona quién los lleva puestos, ya que ofrece información

en de tiempo real. Un dispositivo "*wearable*" es básicamente un dispositivo con un sensor, un procesador, una memoria y capacidades de comunicación, ya sea vía cable o inalámbricamente [3],[4].

La información proporcionada al deportista mediante estos dispositivos es muy importante ya que al estar constantemente monitorizando los datos se puede saber cuándo detener o modular la actividad deportiva para evitar lesiones musculares. O también en caso contrario, saber en qué actividad se puede aumentar el rendimiento.

II. Objetivos

El objetivo principal de este proyecto es el de monitorizar la actividad muscular a partir de señales bioeléctricas asociadas. Estas señales son significativas para los deportistas ya que con una adecuada interpretación podrán facilitarle el estado del sistema muscular en cuestión y saber si debe detener la actividad para evitar fatigas o lesiones musculares. Esto permite modular el entrenamiento de la persona que haga uso de estos sistemas.

Desde un punto de vista tecnológico este proyecto se basa en la integración electrónica de un sistema de adquisición de señales bioeléctricas empleando para ello una tarjeta de adquisición de señales bioeléctricas y un microprocesador ARM Cortex-M3. Es necesario desarrollar el firmware que permite tratar digitalmente estas señales.

Teniendo identificado el objetivo principal, dividimos el mismo en objetivos parciales. Estos objetivos son los siguientes:

1. Estudiar fisiológicamente el músculo y las señales que puede generar.
2. Identificar el mejor método de adquisición de señales musculares.
3. Evaluar el microprocesador ARM Cortex-M3 para su uso sobre el tratamiento de las señales musculares.
4. Implementar la comunicación entre la tarjeta de adquisición y el microprocesador ARM Cortex-M3.
5. Desarrollar el firmware para tratar las señales de tal manera que se obtenga una salida deseada por el display LCD.

6. Desarrollar una aplicación Android sobre la cual monitorizar datos.
7. Comunicar dicha plataforma Android con el microprocesador ARM Cortex-M3.
8. Desarrollar interfaz gráfica de visualización de datos en la plataforma Android.

III. Contenido de la memoria

A continuación se describe brevemente los temas a desarrollar a lo largo de esta memoria del TFG. Se ha dividido la memoria en los siguientes capítulos:

- **Capítulo I: El músculo:** En este capítulo se describe tanto el músculo como sus propiedades. Además se detalla el análisis de las señales bioléctricas conocida como EMG.
- **Capítulo II: Hardware del sistema:** En este capítulo se detalla los criterios de selección de los distintos componentes hardware empleados a lo largo del desarrollo y una descripción de los mismos.
- **Capítulo III: Desarrollo del sistema:** En este capítulo se describirá la integración electrónica adoptada entre los distintos componentes del sistema para la adquisición de la señal bioeléctrica de interés y los distintos procesos que se han tomado para la correcta representación de la misma. También se describe el firmware desarrollado para alcanzar todas las etapas.
- **Capítulo IV: Resultados:** En este capítulo se mostrarán los resultados obtenidos del sistema desarrollado mediante imágenes que faciliten su lectura y análisis.
- **Capítulo V: Conclusiones:** En este capítulo se discutirán las distintas aplicaciones que podría tener el sistema desarrollado además de las posibles líneas futuras que podría presentar.
- **Capítulo VI: Presupuesto:** En este capítulo se detallará el presupuesto correspondiente a la ejecución de este proyecto.

1. El sistema muscular

1.1 Introducción

Una de las características de los animales es su capacidad de realizar movimientos. Este movimiento es posible a la existencia de los músculos, formados por células que pueden modificar su longitud [5].

El cuerpo humano cuenta con un sistema muscular, que es el conjunto de más de 600 músculos [6]. La función mayoritaria de estos músculos es la de producir movimientos en las distintas partes del cuerpo.

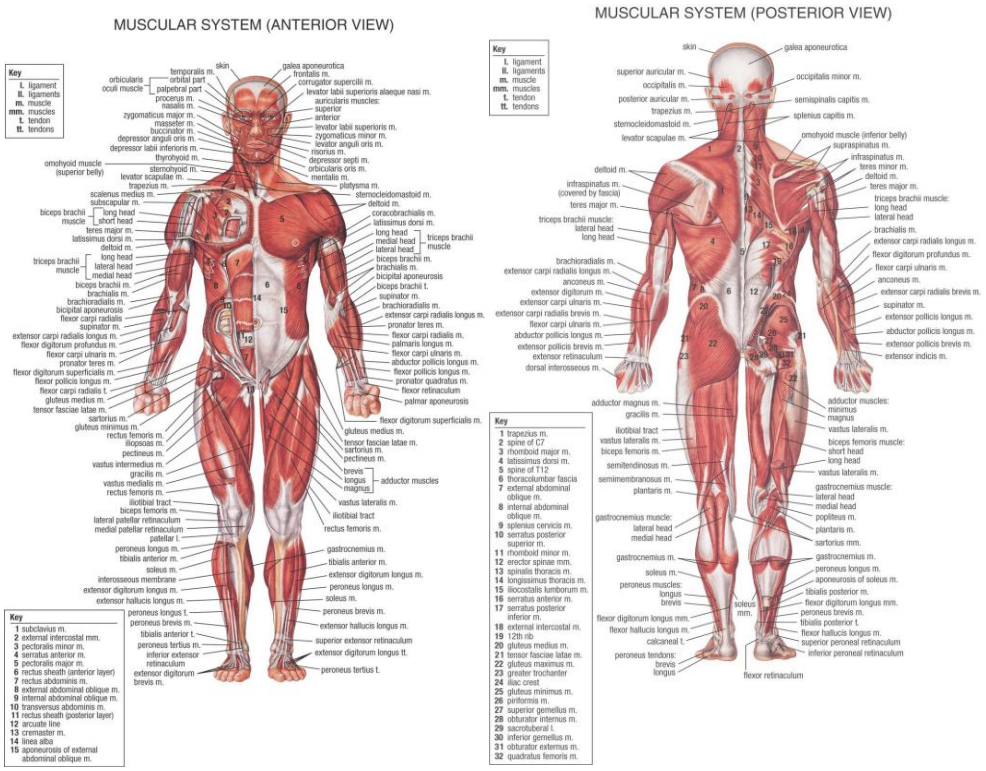

Figura 1. Sistema muscular humano

Existen tres tipos de tejidos musculares, que a su vez conforman tres tipos de músculos:

1. Tejido muscular esquelético: También puede denominarse músculo voluntario o estriado. Se denomina voluntario debido a que se contrae por voluntad propia.

2. Tejido muscular liso: También puede denominarse visceral o involuntario. La localización de estos músculos se encuentra en las paredes de los vasos sanguíneos y linfáticos, el tubo digestivo, las vías respiratorias, la vejiga, las vías biliares y el útero.

3. Tejido muscular cardiaco: Es el músculo que se encuentra en las paredes del corazón. No está bajo el control voluntario [7].

En este proyecto, el músculo de estudio es el denominado músculo esquelético o voluntario, ya que son los músculos que se utilizan cuando realizamos actividades deportivas.

1.2 Músculo esquelético

Son los músculos unidos a los huesos, y son los encargados de hacer que estos se muevan. El músculo es el elemento activo del movimiento, mientras que el hueso es el elemento pasivo del movimiento.

Este tejido muscular es también conocido como estriado voluntario debido a las estrías transversales que este presenta a niveles microscópicos. Es el músculo más abundante del cuerpo humano, con una proporción del 40 % del peso corporal total aproximadamente [8].

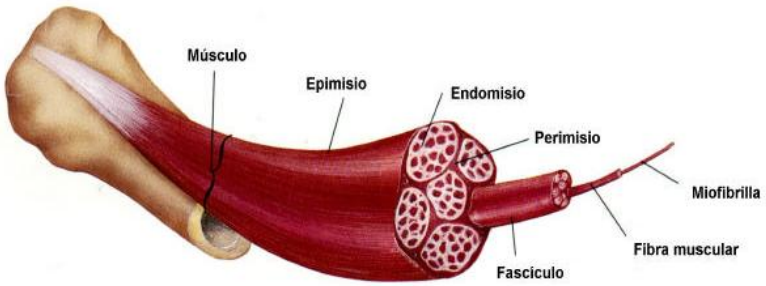

Figura 2. Estructura básica del músculo (adaptado de Wilmore y Costill, 1994)

Los músculos esqueléticos permiten caminar, correr, saltar y otra multitud de acciones voluntarias. Estos músculos responden a impulsos nerviosos, que viajan a través de los nervios motores que terminan en los músculos. Las respuestas del músculo a estos impulsos nerviosos pueden adaptar una u otra forma y además generan señales bioeléctricas.

1.3 Propiedades musculares

A continuación se detallarán unas definiciones de interés asociada a la propiedad de los músculos:

- *Excitabilidad:* Es la facultad de percibir un estímulo y responder al mismo. La respuesta de la fibra muscular es la producción a lo largo de membrana de una corriente eléctrica que origina la contracción muscular.

- *Contractibilidad:* Es la capacidad de contraerse con fuerza ante el estímulo apropiado.

- *Elasticidad:* Es la capacidad que tienen las fibras musculares para acortarse y recuperar su longitud de descanso, después del estiramiento.

- *Extensibilidad:* Es la facultad del estiramiento. Si cuando las fibras musculares se contraen, se acortan, cuando se relajan, pueden estirarse más allá de la longitud de descanso.

- *Plasticidad:* Es la capacidad que tiene el músculo a modificar su estructura en función del trabajo que desempeñe. Así puede ser que un músculo se puede hacer más resistente o más fuerte [8].

1.4 EMG (Electromiografía)

1.4.1 Introducción

La EMG [9] es una técnica experimental que abarca el desarrollo, el registro y el análisis de señales bioeléctricas. Es un estudio sobre la actividad eléctrica de los músculos esqueléticos. Proporciona información muy útil sobre su estado fisiológico y los nervios que lo activan.

Actualmente la EMG es una técnica de investigación y diagnóstico médico que permite detectar y evaluar enfermedades neurológica-musculares, y también se utiliza cada vez más para la investigación en la ciencia del deporte [10]. Si los nervios involucrados en la generación y transmisión de los comandos de activación muscular están afectados, pueden variar las características de generación de los potenciales de acción (frecuencia, amplitud, entre otras), mientras que si son los músculos los que están afectados, varían las características de transmisión de los impulsos (amplitud, velocidad de conducción, entre otras) [11].

Las señales bioeléctricas se miden mediante dispositivos llamados electrodos, los cuales se encargan de convertir las corrientes de tipo iónico producidas por la distribución de potencial generada en el interior del tejido vivo en corrientes de tipo eléctrico que pueden ser medidas y acondicionadas para su posterior análisis y tratamiento. Este estudio se realiza con electrodos superficiales, que son pequeños electrodos o discos que se adhieren a la piel. Esta técnica se conoce como SEMG [12], *Surface EMG*, EMG superficial. La EMG superficial es el método más común de medida, puesto que es no invasiva y puede ser realizada con un mínimo de riesgo sobre el paciente.

1.4.2 Propiedades bioeléctricas del tejido muscular esquelético

Cuando la membrana de los músculos es excitada mediante la corriente iónica generada por los axonas (prolongación de neuronas especializadas en conducir el impulso nervioso desde el cuerpo celular hacia otra célula) de las neuronas motoras, cambia la permeabilidad selectiva permitiendo la entrada de iones a las células, de manera que se genera una avalancha de dichos iones al interior de la célula para intentar reestablecer el equilibrio, con lo que el potencial de la transmembrana resulta ligeramente positivo. Este potencial de acción es el que se conoce como potencial de acción (PA) de la unidad motora (UM), (PAUM), y se dice que en ese momento, las fibras musculares se encuentran despolarizadas. El proceso por el cual las mismas cambian de estado de reposo a estado de acción se denomina despolarización y es seguido de un evento de repolarización que retorna a las fibras al estado de reposo [13]. Esta actividad se puede observar en la Figura 3.

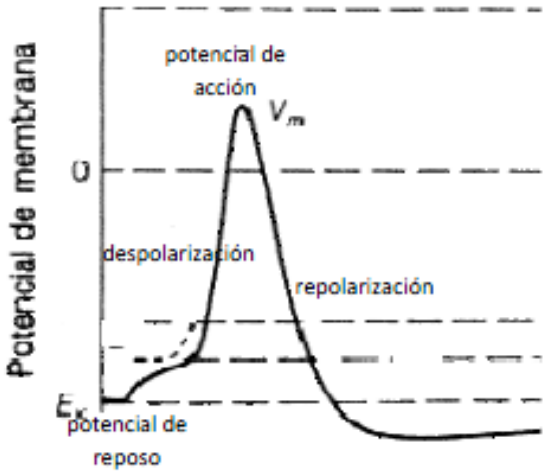

Figura 3. Transición entre el potencial de reposo y el potencial de acción

Las fibras musculares (FMs) son células alternantes debido a las proteínas contráctiles presentes en su interior. La contracción se verifica mediante el desplazamiento de estas proteínas entre sí. Es el sistema nervioso quien codifica el grado de contracción de las fibras musculares (FMs) según la frecuencia de los impulsos nerviosos de las motoneuronas alfa (neurona del sistema nervioso central que proyecta

su axón hacia un músculo o glándula). Los impulsos nerviosos son potenciales de acción (PAs) que se transmiten a las células musculares.

El conjunto que forma una motoneurona alfa y las FMs inervadas por ella se conoce como unidad motora (UM) y constituye la unidad anatómica y funcional del músculo.

El registro de las descargas de las fibras musculares (FMs) de una unidad motora (UM) se conoce como potenciales de acción (PA) de unidad motora (PAUM). En condiciones normales, la amplitud media de los PAUMs es de unos 0,5 mV y la duración varía entre 8 y 14 ms según el tamaño de las UMs.

La amplitud, y las propiedades de las señales EMG tanto en el dominio del tiempo como en la frecuencia dependen de factores tales como:

- El tiempo y la intensidad de la contracción muscular.
- La distancia entre el electrodo y la zona de actividad muscular.
- Las propiedades de la piel (por ejemplo el espesor de la piel y tejido adiposo).
- Las propiedades del electrodo y el amplificador.
- La calidad del contacto entre la piel y el electrodo.

La exploración EMG comprende el registro de la señal bioeléctrica en tres estados funcionales.

- *En reposo*: No se encuentra actividad debida a la relajación del músculo.
- *Durante una contracción débil*: Se activan un número escaso de UMs y se pueden captar las PAUMs correspondientes. Para lograr esto el paciente debe realizar una contracción débil y mantenida. Si el grado de contracción es excesivo se activan demasiadas UMs y las descargas se superponen, con lo que sus formas de onda (FOs) se distorsionan.
- *Durante una contracción voluntaria máxima*: En este estado se genera información sobre la población de UMs funcionantes.

1.4.3 Características de las señales EMG

Una de las características básicas para lograr los objetivos de este proyecto es saber el rango espectral de las señales EMG. La frecuencia de las señales EMG generales comprende un rango aproximado de entre 0 y 500 Hz, pero como en este trabajo se van a adquirir las señales a través de electrodos superficiales, reduce este rango espectral a una banda entre 10 y 400 Hz. Dicho ancho de banda se ve afectado principalmente por la forma de onda (FO) de los PAUMs de las UMs activas, aunque también depende de la frecuencia de generación de los PAUMs. No obstante, tanto por el método de adquisición con electrodos superficiales o electrodos de inserción, la energía espectral se concentra en los rangos de 50 a 150 Hz. Obsérvese dicho fenómeno en la Figura 4.

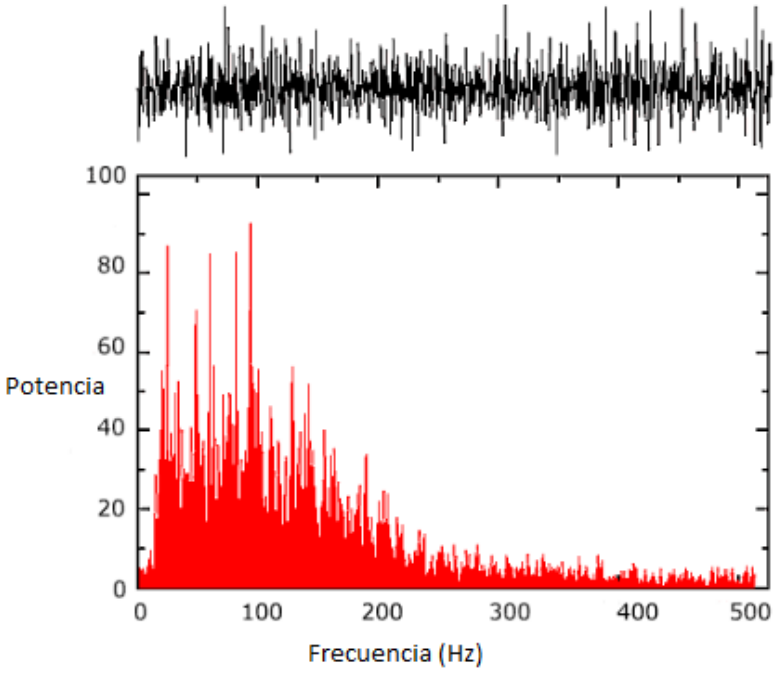

Figura 4. Rango espectral de las señales EMG

Otra característica importante es la amplitud, que depende del número de UM reclutadas, su profundidad y un número de fibras que componen cada una de ellas. Al igual que también depende de la frecuencia con la que son activadas [14].

La característica común para la clasificación de la señal EMG, es la distancia euclidiana (norma característica de la geometría euclidiana que indica cuán lejos está un punto de otro en el plano o el espacio) entre la forma de onda de los PAUM. Las

principales características de la señal EMG son el número de unidades motoras activas, la forma de onda de PAUM y el tiempo de inervación. En el estudio de Wellig y Moschytz, se determinó que la forma de onda de un PAUM y el número de unidades motoras activas en una contracción muscular conforman un problema cuando se desea clasificar las EMGs [15].

1.4.4 Procesos para la adquisición de una señal EMG

Hay una serie de pasos a seguir para conseguir una señal EMG [16] de forma que los expertos puedan interpretar lo que dicha señal significa. Para ello, el primer paso a ejecutar es el de la colocación de los electrodos superficiales. Los electrodos superficiales constan, normalmente de tres electrodos. Dos electrodos activos y un electrodo de referencia (mayormente negro).

Se presentan a continuación las técnicas básicas de procesamiento de la señal electromiográfica:

- *Colocación electrodos:* Lo primero es identificar el músculo a medir. Se recomienda que la piel sobre la cual se van a habituar dichos electrodos esté limpia y seca. En algunos casos es incluso favorable afeitarse. A continuación se procede a la colocación de los electrodos. Los electrodos activos se colocan sobre el músculo de tal manera que estén alineados con las fibras musculares. Debe haber un electrodo, en medida de lo posible, lo más céntrico posible (en referencia al músculo) y el otro con una pequeña separación tirando hacia el final del músculo. El electrodo de referencia se coloca preferentemente un poco más alejado de ambos electrodos activos, y si es posible en algún hueso.

- *Amplificación:* Toda la instrumentación va dirigida a obtener una representación inteligible de los PAs musculares. Para ello los sistemas deben ser altamente sensibles porque las magnitudes bioeléctricas son muy pequeñas. De ahí que sea necesario amplificar la señal entre 34 y 108 dB [11].

- *Eliminación de ruido:* El ruido, ya sea origen técnico o biológico, es un parámetro indeseado que acompaña a la señal que se pretende registrar y generalmente supera la magnitud de la misma. Los amplificadores diferenciales son capaces de eliminar el ruido debido a otros músculos (por ejemplo, el corazón), ya que son capaces de medir la diferencia de potencial entre los electrodos activos y de referencia, como se puede observar en la Figura 5 [11].

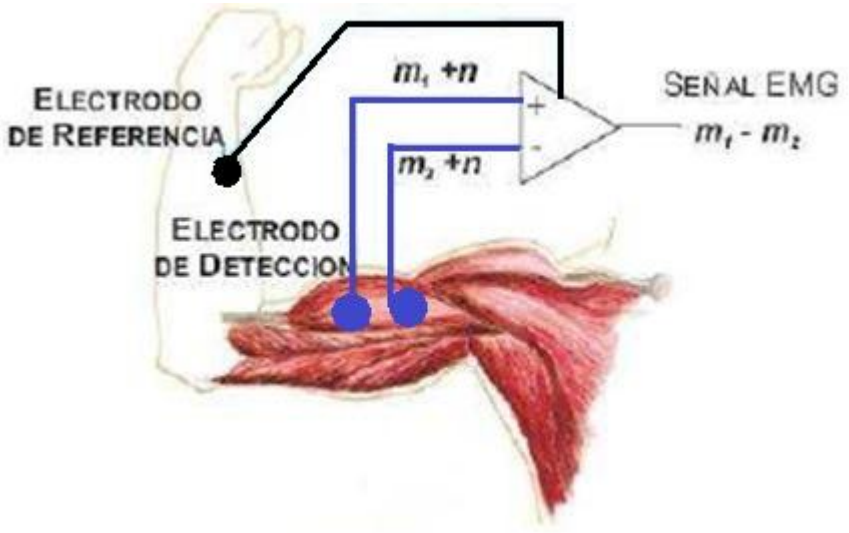

Figura 5. Esquemático de la configuración del amplificador diferencial. La señal EMG representada como `m´y ruido como `n´

- *Filtrado:* Se recomienda eliminar frecuencias inferiores a 2-5 Hz (fluctuaciones lentas por movimiento de aguja si estamos en medidas invasivas) y superiores a 10 kHz (oscilaciones de la señal de origen técnico). Además se recomienda eliminar la señal a 50 Hz, siempre y cuando se está conectado a la corriente alterna de la red eléctrica, en "wearables" no hace falta [11].

- Digitalización: Este proceso, ejecutado en convertidores analógicos-digitales, consiste en la obtención de medidas (muestras) a intervalos regulares de tiempo. La frecuencia de muestreo de tal conversor debe ser lo suficientemente alta como para no perder cambios significativos del voltaje [11]. La tasa de muestreo en la práctica puede ser de 1000 Hz con el fin de registrar componentes de frecuencia en la señal hasta los 500

Hz, según el teorema de Nyquist. No se requiere de mayores tasas, puesto que la mayor concentración de energía se encuentra entre los 50 y 150 Hz [17].

1.4.5 Campos de aplicación

Las aplicaciones de la electromiografía siempre han tomado una dirección paralela a la medicina [18]. Ya sea como técnica de diagnóstico para detectar enfermedades o como dispositivo para detectar anomalías musculares. La EMG ha significado un gran avance en el estudio de la fisiología muscular.

La capacidad de detectar anomalías es la que ha introducido a la EMG en el ámbito del deporte. Existe un campo extenso de aplicaciones deportivas para sistemas que contienen el análisis de una electromiografía, ya que sirven como guía para un deportista sabiendo si sus músculos en cuestión están aptos para el ejercicio físico y, sobre todo, si puede dar más de sí.

1.5 Solución propuesta

La solución propuesta consta de un sistema capaz de adquirir señales electromiográficas, que son procesadas y posteriormente mostradas por el display de la tarjeta de desarrollo. Paralelamente se desarrolló una aplicación en Android donde se pueden visualizar las mismas señales procesadas.

Para la realización de este sistema se han utilizado sistemas independientes, es decir, la parte de adquisición y la de procesado son dos sistemas distintos.

A continuación se mostrará en la Figura 6 el diagrama de bloques del conjunto de los sistemas como sistema unificado.

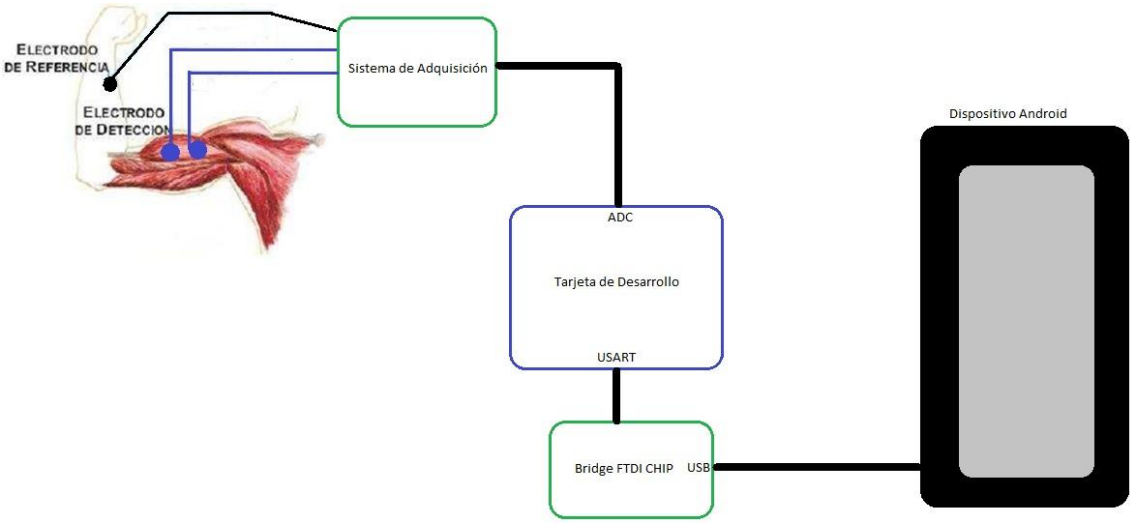

Figura 6. Sistema total propuesto

Como se puede observar en la Figura 6 el sistema está compuesto por distintos subsistemas.

El primer bloque es el denominado como sistema de adquisición. Este bloque consta de una tarjeta de acondicionamiento de señal muscular cuya tarjeta es la denominada Muscle Sensor v3. Esta tarjeta consigue adquirir las señales bioeléctricas del músculo sobre el cual se posicionan los electrodos. La señal proporcionada es enviada hacia el microcontrolador para su procesado.

El segundo bloque consta de la tarjeta de desarrollo. En esta tarjeta se encuentra el microcontralador y el display LCD. En este sistema la tarea es la de procesar la señal para conseguir una representación de la señal bioeléctrica en la pantalla del LCD. La señal es adquirida por el microprocesador a través de un canal de su convertidor analógico/digital. La comunicación entre el microprocesador y la pantalla es vía SPI.

Y por último, el tercer bloque. Este último bloque se forma con el Bridge FTDI CHIP y un dispositivo Android. La tarea del FTDI CHIP es el de hacer como puente de comunicación entre el segundo bloque y el dispositivo Android. El dispositivo Android muestra en la aplicación creada las gráficas correspondientes a la actividad muscular.

2. Hardware del sistema

2.1 Introducción

En este capítulo se describirán todos aquellos dispositivos hardware que se hayan empleado durante el desarrollo de TFG. Se tratará de dar a entender que aportan sus especificaciones y prestaciones al proyecto. En algunos casos, como el del microprocesador, no se comentarán todas sus cualidades, se hará hincapié en los servicios que se han utilizado para la realización del TFG.

2.2 Muscle Sensor v3

2.2.1 Introducción

El Muscle Sensor v3 es una tarjeta de acondicionamiento de señales musculares. Su principal función es la de generar una señal rectificada, filtrada y amplificada a la salida de sus pines. En este dispositivo el procesado que se realiza sobre la señal es analógico.

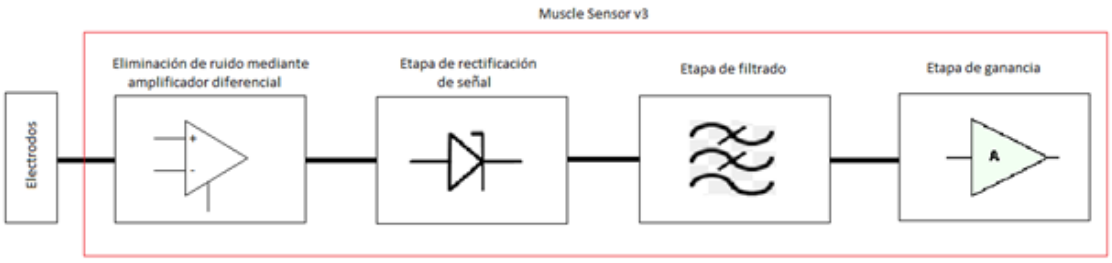

Figura 7. Diagrama de bloques del Muscle Sensor v3

En la Figura 7 se observa el diagrama de bloques de las distintas etapas que atraviesa la señal bioeléctrica en esta tarjeta. En la Figura 8 se observa la tarjeta Muscle Sensor v3.

Figura 8. Tarjeta Muscle Sensor v3

A parte de esta tarjeta, se necesitan los electrodos que irán colocados sobre el músculo en cuestión. Estos parches se comunican con el sensor a través de un conector Jack como se indica en la Figura 9.

Figura 9. Parches y conector Jack para el Muscle Sensor v3

2.2.2 Características

El Muscle Sensor v3 cuenta con unas dimensiones adecuadas para trabajar con dispositivos "wearables". A continuación en la Figura 10 se pueden ver las dimensiones reales.

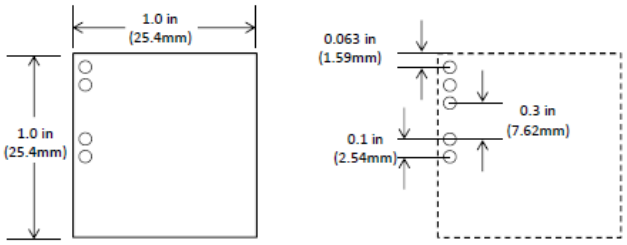

Figura 10. Dimensiones Muscle Sensor v3

Otras características que ofrece el Muscle Sensor v3 son las que se enumeran a continuación:

- Especialmente diseñado para trabajar con microprocesadores.
- Posee una ganancia ajustable.
- Colocación de pines según el estándar *breadboard* (separación de pines como en una protoboard).

A la salida se tiene una señal EMG amplificada, recitifcada y filtrada a efectos de que se pueda conectar con el convertidor analógico-digital de cualquier microprocesador.

2.2.3 Especificaciones eléctricas

A continuación se mostrarán mediante una tabla las especificaciones eléctricas que nos facilita el fabricante [19].

Tabla 1. Especificaciones eléctricas del Muscle Sensor v3

Parámetros	Mínimo	Típico	Máximo
Alimentación (Vs)	±3V	±5V	±30V
Ganancia ajustable	0.01 Ω	50 kΩ	100 kΩ
Tensión señal de Salida	0 V	---	+ Vs
Tensión de entrada diferencial	0 mV	2–5 mV	+Vs/Ganancia

2.2.4 Esquemático

En este apartado se mostrará el esquemático del Muscle Sensor v3 donde se muestran los circuitos que atraviesa la señal desde la entrada hasta la salida.

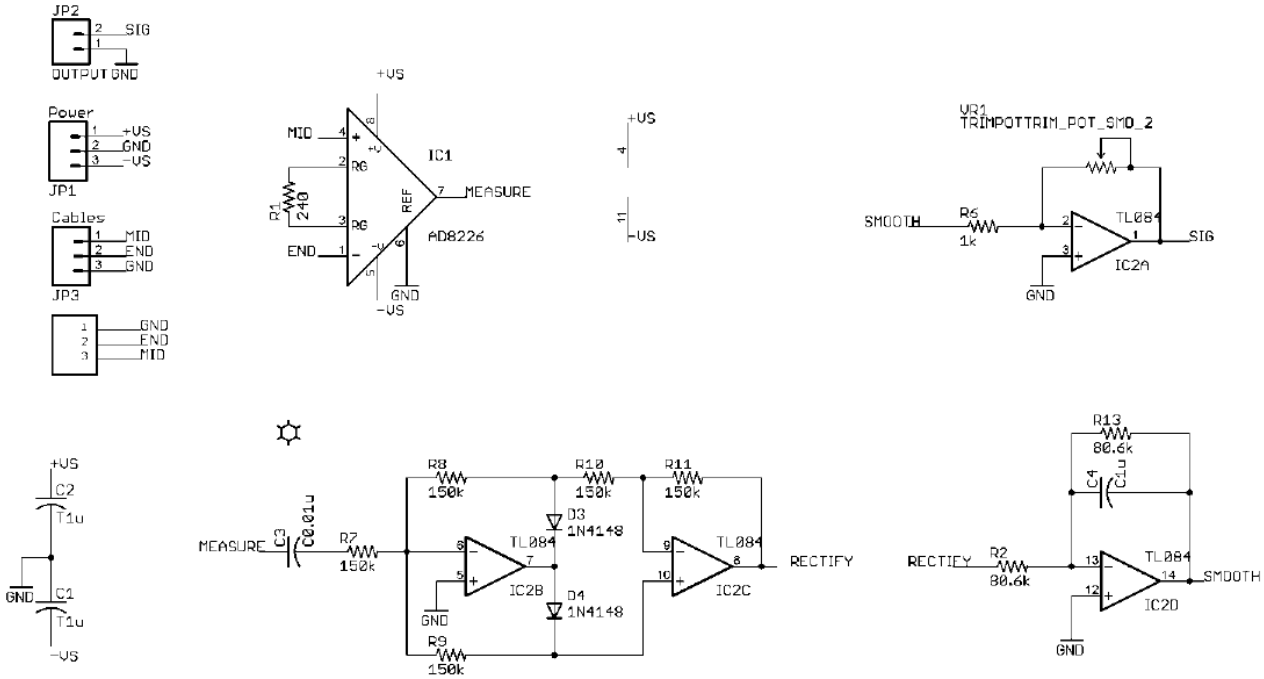

Figura 11. Esquemático del Muscle Sensor v3

En el esquemático se pueden diferenciar claramente las tres etapas que se han estado nombrando hasta ahora.

Disponemos del primer circuito por el que pasa la señal. La etapa de amplificación, que consta de un amplificador diferencial.

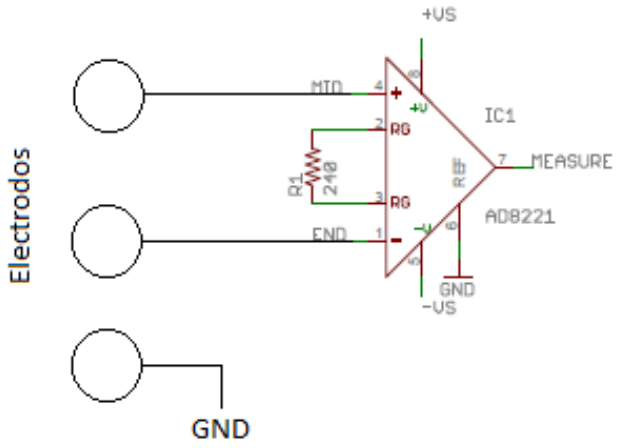

Figura 12. Etapa de amplificación Muscle Sensor v3

El amplificador diferencial utilizado en el Muscle Sensor v3 en la etapa de eliminación de ruido es el AD8221. El AD8221 [20] es un amplificador de

instrumentación de ganancia programable y un alto rendimiento que ofrece uno de los mayores CMRR (Razón de Rechazo al Modo Común) para frecuencias altas. El CMRR de amplificadores de instrumentación que se encuentran en mercados actualmente decaen sobre los 200 Hz. Sin embargo, el AD8221 mantiene un mínimo de 80 dB de CMRR para una frecuencia de 10 kHz. En la Figura 13 se observa el esquemático simplificado del AD8221.

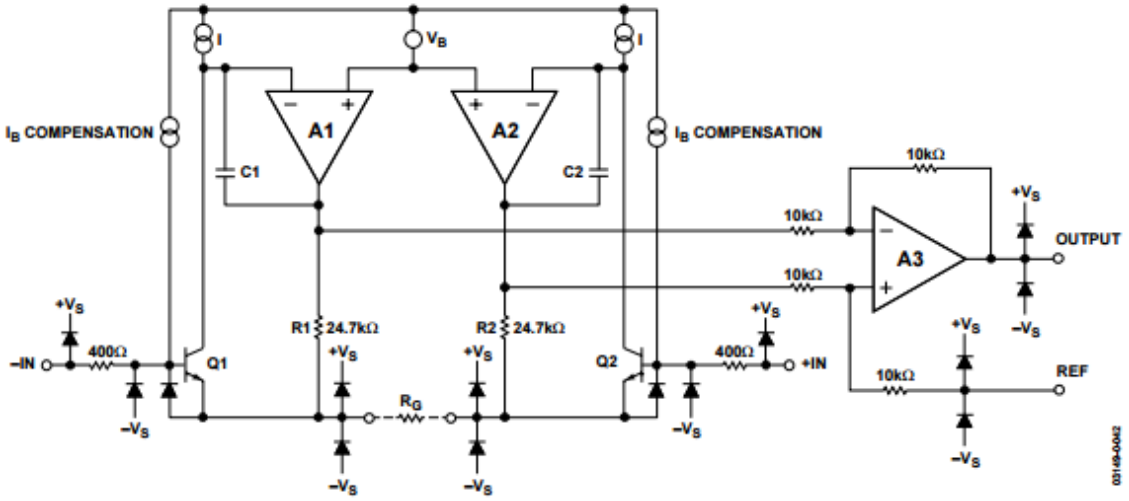

Figura 13. Diagrama interno del AD8221

El AD8221 es un amplificador de instrumentación basado en la topología típica de 3 amplificadores operacionales. Los transistores de entrada Q1 y Q2 asociadas a corrientes fijas, de manera que cualquier señal diferencial a la entrada forzará que los voltajes de salida de A1 y A2 cambien como corresponde. Una señal aplicada a la entrada crea una corriente que pasa a través de R_G, R_1 y R_2, por lo que las salidas de A1 y A2 proporcionan el voltaje correspondiente. Las señales diferenciales y la de modo común son aplicadas a un amplificador diferencial que rechaza voltaje a modo común pero amplifica el voltaje diferencial. La función de transferencia del AD8221 es:

$$G = 1 + \frac{49.4\ k\Omega}{R_G}$$

Las especificaciones máximas capaces de soportar este amplificador operacional son las que se detallan a continuación.

Parameter	Rating
Supply Voltage	±18 V
Internal Power Dissipation	200 mW
Output Short Circuit Current	Indefinite
Input Voltage (Common-Mode)	$\pm V_S$
Differential Input Voltage	±Vs
Storage Temperature	−65°C to +150°C
Operational* Temperature Range	−40°C to +125°C

Figura 14. Especificaciones máximas

El AD8221 es idóneo para aplicaciones que requieren una precisa adquisición de datos, para aplicaciones de análisis biomédico o para aplicaciones de instrumentación aeroespacial.

La segunda etapa que atraviesa la señal, por lo que sigue al amplificador, es el circuito rectificador de la señal, cuyo propósito es conseguir que todos los valores a su salida tengan carácter positivo (por encima de 0).

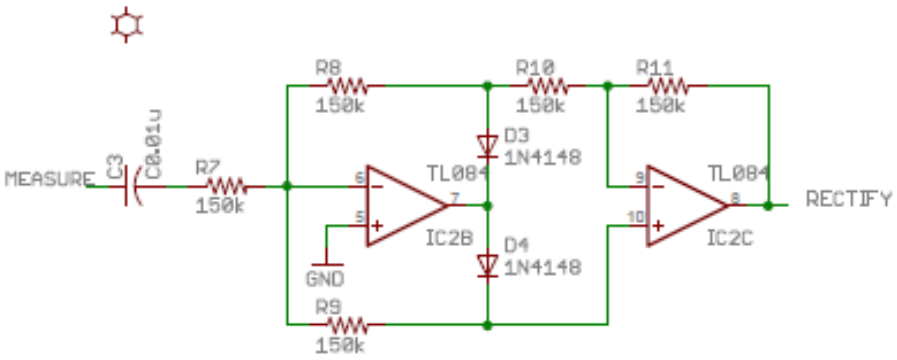

Figura 15. Etapa de rectificación de señal del Muscle Sensor v3

Los rectificadores de precisión convencionales tienen el problema al rectificar tensiones pequeñas (por ejemplo, 0.1 V). La caída de tensión introduce un error muy significativo en el proceso. El uso de rectificadores basados en amplificadores operacionales permite solventar este problema, pudiendo rectificar señales de muy baja amplitud sin problema [21]. El circuito de la Figura 15 es un rectificador precisión de onda completa con resistencias iguales basado en amplificadores operacionales [22].

Todas las resistencias de este circuito son iguales y su impedancia de entrada es igual al valor de la resistencia. En la Figura 16 se muestran las direcciones de corriente (flechas negras) para señales de entrada positivas. El diodo D_3 conduce, de manera que

los amplificadores operacionales funcionan como inversores y la señal de salida de esta etapa es igual al de la entrada.

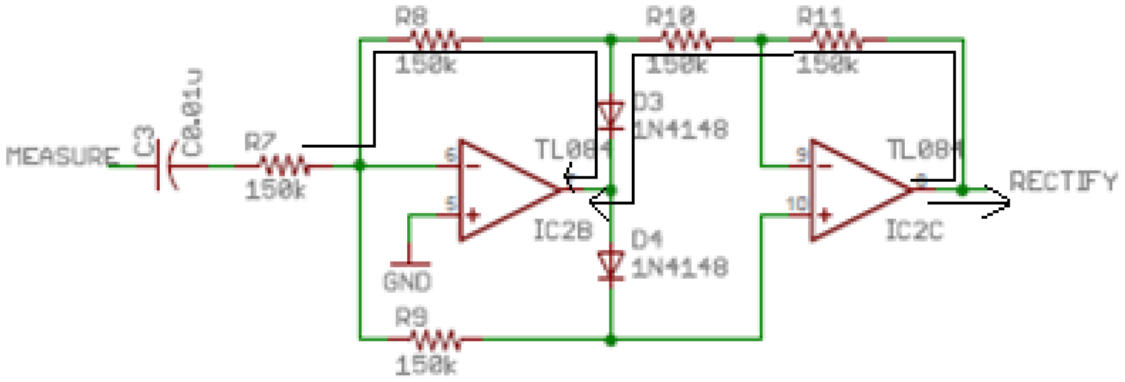

Figura 16. Sentido de corriente para señales positivas

Cuando las señales de entrada son negativas, el diodo que conduce es el D_4, por lo que el amplificador operacional TL084 IC2B funciona como inversor. Entonces, el voltaje de salida es positivo e igual al valor absoluto de la señal de entrada.

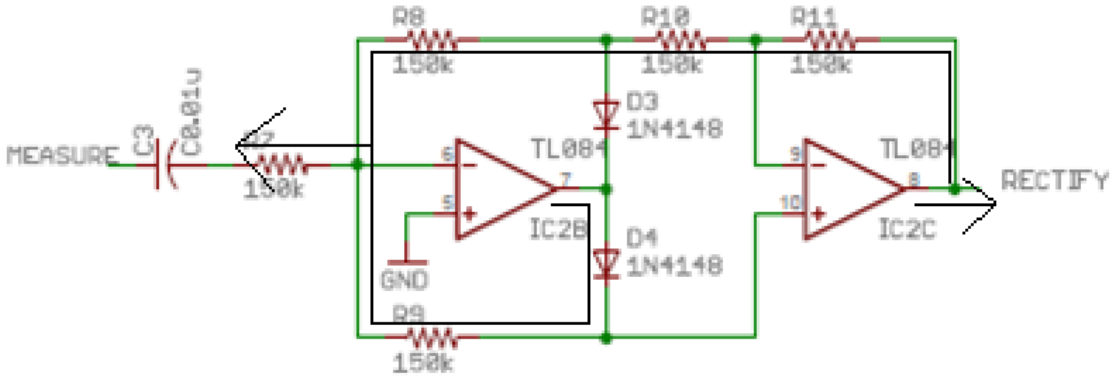

Figura 17. Sentido de corriente para señales negativas

El circuito siguiente a este, es el de filtrado paso bajo de señal. El circuito que se encarga de esto es el siguiente.

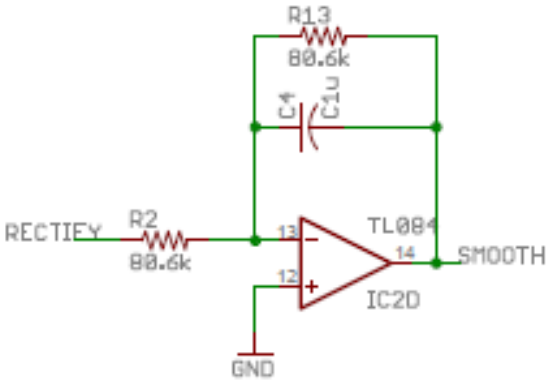

Figura 18. Etapa de "suavizado" de señal el Muscle Sensor v3

De la Figura 18 se puede asumir que se trata de un filtro RC paso bajo. Para calcular la frecuencia de corte del filtro se usa la siguiente expresión:

$$\omega_o = \frac{1}{R \cdot C} = \frac{1}{80.6\ k\Omega \cdot 1\ \mu F} = 12.4069\ rad/seg$$

$$f_o = \frac{\omega_o}{2\pi} = 1.9746\ Hz$$

Esta etapa de filtrado tiene como fin reducir la cantidad de variaciones de amplitud que hay en la señal. Elimina aquellos valores cuyos son muy distintos al de sus vecinos. El efecto que tiene sobre una señal este tipo de filtrado es el que se muestra en la Figura 19, donde se toma como ejemplo una señal sinusoidal rectificada.

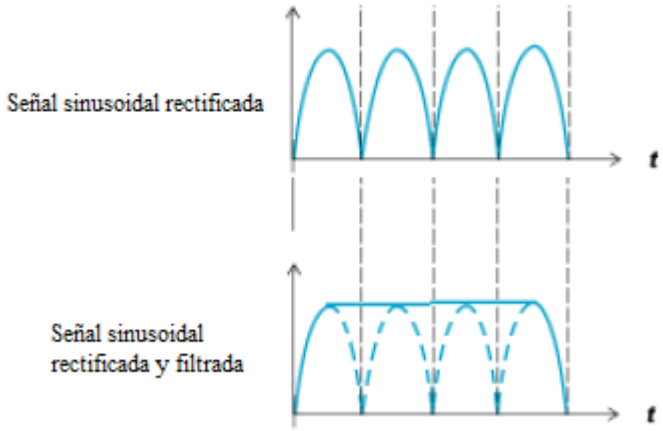

Figura 19. Efecto etapa de filtrado sobre una señal sinusoidal

La configuración del PIN LAYOUT se muestra en la Figura 20.

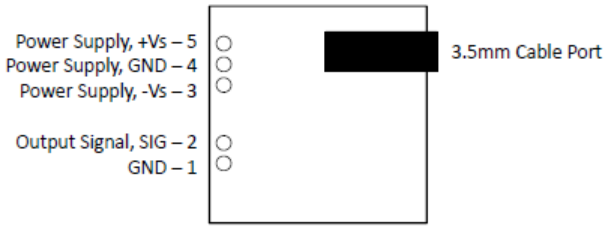

Figura 20. Pin Layout del Muscle Sensor v3

Se ha optado por esta tarjeta de acondicionamiento, además de su precio económico, porque ofrece las prestaciones necesarias para el desarrollo del TFG.

2.3 Microprocesador

2.3.1 Introducción

Para el desarrollo de este trabajo es importante seleccionar el componente sobre el que recae la mayor parte de carga computacional del sistema, y que permite integrar los distintos periféricos para que realicen una función conjunta. Para este TFG se ha optado por un microprocesador de ST Microelectronics de la familia STM32, su modelo es STM32F103VCT6 [23], basado en ARM Cortex-M3.

2.3.2 ARM Cortex-M3

La familia ARM Cortex es una nueva generación de procesadores que proporcionan una arquitectura estándar para un amplio rango de demandas tecnológicas.

Existen tres perfiles de Cortex: Un perfil A para aplicaciones de alto rendimiento, un perfil R para aplicaciones de tiempo real y un perfil M para aplicaciones con relación de bajo costo y buenas prestaciones del microprocesador.

El ARM Cortex-M3 [24] es el procesador líder en la industria de los procesadores de 32 bit para aplicaciones altamente deterministas y de tiempo real,

desarrollado para permitir a las empresas desarrollar plataformas de alto rendimiento a bajo costo. El procesador ofrece un excelente rendimiento computacional y una sobresaliente respuesta a eventos. Además es altamente configurable permitiendo un amplio rango de implementaciones que requieran protección de memoria y tecnología potente para dispositivos que requieren un área mínimo.

Como en el ARM9 y ARM7, el Cortex-M3 [25] tiene un pipeline de 3 estados (Fetch, Decode y Execute).

El modelo de programación del Cortex-M3 trata de una arquitectura load/store, donde los datos primero han de moverse sobre un registro para luego ejercer una acción sobre ellos.

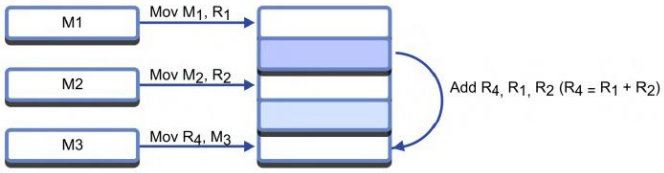

Figura 21. Ejemplo modelo de programación ARM Cortex-M3

El Cortex-M3 dispone de 16 registros. Los registros del R0-R12 son de propósito general, por ejemplo, guardar variables y estados. Además posee unos registros "especiales". Éstos son el R13, el cual hace referencia al Stack Pointer. El R14 es el Link Register, que indica la dirección de retorno, muy útil cuando se producen llamadas a subrutinas. Y por último el R15, que es el Program Counter, el que indica en cada momento a la CPU la siguiente línea de código a ejecutar.

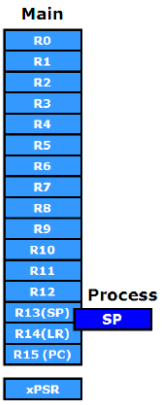

Figura 22. Registros ARM Cortex-M3

El xPSR es un registro conocido como Program Status Register. Este registro contiene algunos campos de bits que influyen en la ejecución de instrucciones de la CPU. Por ejemplo, uno de estos campos son los flags de condiciones como 'N' (Negative, 'Z' (Zero), 'C' (Carry) y 'V' (Overflow) que se activaran dependiendo del resultado de cada instrucción.

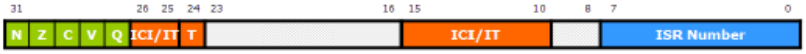

Figura 23. Registro xPSR ARM Cortex-M3

En cuanto al modo de operación, el Cortex-M3 puede ser usado en dos tipos de modos. Estos dos modos son:

- Handler (Supervisor): Donde se tiene un acceso privilegiado a la memoria y recursos de la CPU.
- Thread (Usuario): Se puede estar en modo privilegiado o no. El modo privilegiado funciona muy parecido al modo Handler, y el modo no privilegiado hay zonas restringidas a donde no se puede acceder.

		Operations (privilege out of reset)	Stacks (Main out of reset)
Modes (Thread out of reset)	**Handler** - An exception is being processed	Privileged execution Full control	Main Stack Used by OS and Exceptions
	Thread - No exception is being processed - Normal code is executing	Privileged/Unprivileged	Main/Process

Figura 24. Modos de operación del ARM Cortex-M3

Cabe destacar que la CPU del Cortex está diseñada para ejecutar instrucciones de tipo Thumb-2, que son instrucciones de 16 y 32 bits. El Thumb-2 mejora en un 26% la densidad del código y en un 25% el rendimiento frente a instrucciones Thumb de 16 bits.

Y por último comentar que el procesador ARM Cortex-M3 integra un controlador de interrupciones anidado (Nested Vector Interrupt Controller) para obtener un alto rendimiento en situaciones que puedan depender de la importancia de la respuesta ante interrupciones. El NVIC incluye una interrupción no enmascarable (NMI), y ofrece hasta 256 canales de prioridad para interrupciones. La integración ajustada entre el procesador y el NVIC permite una ejecución rápida de la rutina de servicio de la interrupción (ISR), reduciendo de manera drástica la latencia ante interrupciones.

En nuestro caso se tiene un STM32, que está basado en un Cortex-M3, especialmente diseñado para aplicaciones de buen rendimiento combinado con un bajo consumo de potencia. El STM32 tiene un direccionamiento posible de 4Gbyte de direcciones bien particionadas para memoria, periféricos y propios periféricos del sistema. Consiste en arquitectura Harvard que además tiene múltiples buses que permitirán operaciones en paralelo incrementando el rendimiento general. La familia Cortex permite además el acceso a datos desalineados, asegurando un uso más eficiente de la memoria SRAM.

La familia STM32 tiene 4 distintos grupos. Éstos son el “Performance Line”, “Access Line” y el “USB Access Line”. Además ha anunciado un cuarto grupo llamado “Connectivity Line”.

El “Access Line” es la entrada para la familia STM32 con operaciones y simples periféricos a 36 MHz. El “Performance Line” trabaja a 72 MHz y proporciona más periféricos. Y el “USB Access Line” añade dispositivos USB para aplicaciones USB relacionadas con costo y prestaciones. El cuarto grupo llamado “Connectivity Line” trae consigo la posibilidad de comunicaciones avanzadas incluyendo un controlador USB de doble rol y un Ethernet MAC [9].

En este caso en concreto se tiene un STM32 de 100 pines, con una memoria Flash de 256 Kbyte de tamaño, es por ello que se concluye que el microprocesador concreto utilizado en el desarrollo del trabajo es el siguiente.

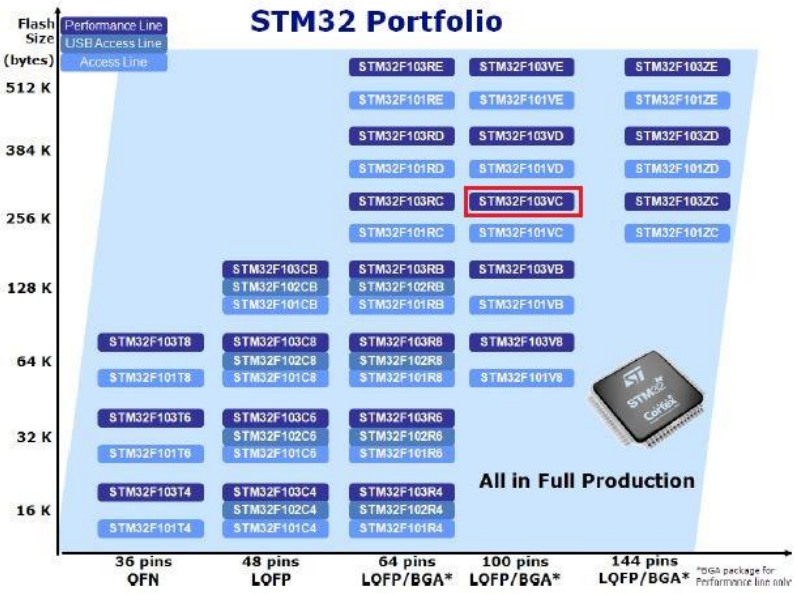

Figura 25. Microprocesador con el que se trabaja

2.3.3 Características del STM32F103VCT6

El STMF32F103VCT6 [26] incorpora un ARM Cortex-M3 de 32 bit tipo RISC de alto rendimiento que opera a una frecuencia de 72 MHz. Este microprocesador pertenece a la familia de microprocesadores STM32F103xC. La diferencia entre unos y otros se basa en el tamaño de memoria, el número de canales del convertidor Analógico/Digital, el número de puertos de entrada/salida, el número de timers y el número de interfaces de comunicación.

Este procesador trabaja en un rango de tensión de alimentación de 2.0 a 3.6 V. Dispone además de un modo de ahorro de energía que permite diseñar aplicaciones de bajo consumo.

Estas características convierten al STMF32F103VCT6 en un microprocesador ajustable a un amplio rango de aplicaciones tales como de control, médicas, industriales, automoción, videojuegos, entre otros.

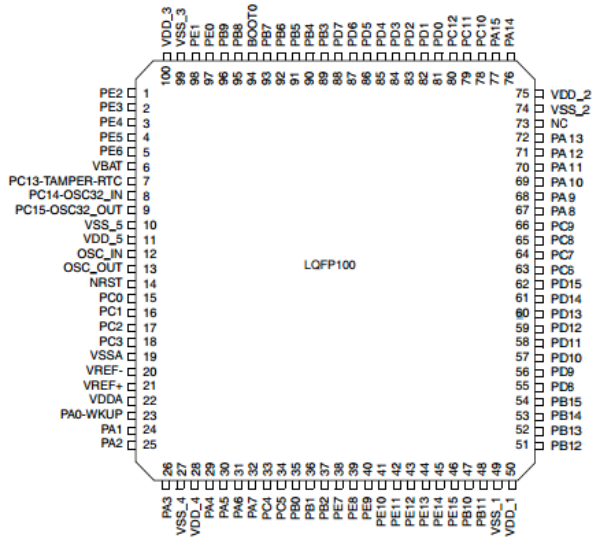

Figura 26. Configuración de pines del STM32F103VCT6

A continuación se exponen algunos de los componentes más relevantes del STM32F103VCT6:

- Frecuencia CPU: 72 MHz.
- Memoria Flash: 256 Kbytes.
- SRAM: 48 Kbytes.
- 8 Timers: 4 de propósito general, 2 de control avanzado y 2 básicos.
- Interfaces de comunicación:
    - 3 SPI, dos de ellos se pueden utilizar como I^2S.
    - 2 I^2C.
    - 5 USART.
    - 1 USB.
    - 1 CAN.
    - 1 SDIO.
- 80 GPIOs.
- 3 ADC de 16 canales.
- 2 DAC (Digital to Analog Converter – Conversor Digital a Analógico) de 2 canales.

2.4 Placa de desarrollo HY-MiniSTM32V

2.4.1 Introducción

En este apartado se describirá la placa de desarrollo HY-MiniSTM32V [27], utilizada para el desarrollo del proyecto. En esta placa viene integrado el procesador STM32F103VCT6 antes descrito. Se trata de una tarjeta de desarrollo de bajo costo y con múltiples prestaciones aptas para el TFG.

A continuación se describirán las características más importantes que ofrece esta tarjeta.

2.4.2 Características

Esta tarjeta de desarrollo consta de muchas prestacione. A continuación se enumeran los componentes de la placa HY-MiniSTM32V.

La tarjeta consta de una pantalla LCD TFT de 3.2", con una resolución de 320x240 píxeles y capacidad de representar hasta 262.000 colores (64 niveles para cada color básico RGB). Es una pantalla táctil resistiva con el controlador RSM1843. El tamaño de la pantalla es de 62x95 mm, y el tamaño completo de la tarjeta es de 80x95 mm.

Dispone de 4 LEDs, 2 de ellos como LEDs indicadores. Dos botones GPIO (KEYA y KEYB), un botón RESET y un botón de Boot-Loader. Incorpora un regulador líneal de 5 V a 3.3 V, una interfaz de depuración JTAG/SWD (20 pines). La tarjeta se alimenta vía USB.

Como es habitual en los microcontroladores actuales, la mayoría de los pines pueden ser programados para que realicen distintas funciones, especialmente los que están agrupados en los cinco puertos (PA, PB, PC, PD y PE), de 16 líneas cada uno. Aparte de los pines utilizados por la propia placa que no están disponibles para el usuario, hay dos de ellos que activan sendos LEDS de uso general y dos que activan a pulsadores también utilizables para cualquier función. Hay 30 pines no conectados y por tanto disponibles y configurables como entradas o salidas de distintos tipos, 16 de ellos pueden además actuar como entradas analógicas con conversor ADC de 12 bits, dos

como salidas analógicas tipo DAC, y también hay un cierto número de salidas de impulsos PWM modulados en duración, temporizadores y muchas otras funciones especiales. Por otra parte, los pines del microcontrolador, con o sin uso, están todos disponibles entre los conectores E/S superior e inferior.

En la Figura 27 se muestra la tarjeta de desarrollo HY-MiniSTM32V. Como se puede observar, en la parte superior de la tarjeta se aprecian dos bases mini-USB, pero sólo uno de ellos ejerce como USB 2.0 real (el de la izquierda). La otra base es en realidad un receptor USB que conduce las señales a un chip convertidor, el PL2303HX concretamente, y se convierten a señales RS-232. Estas señales son luego luego llevadas al microprocesador.

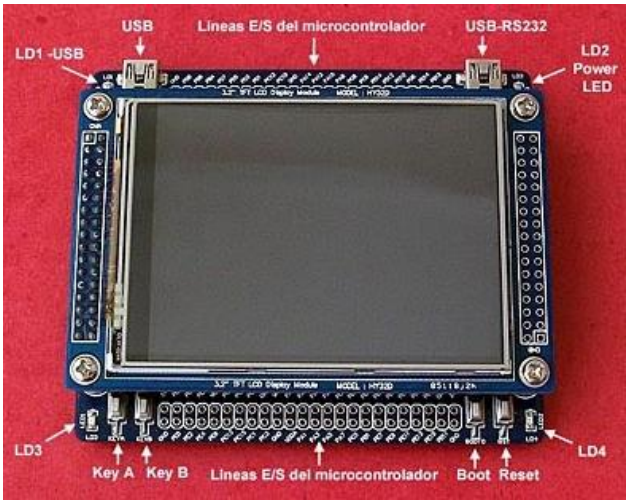

Figura 27. Parte anterior de la placa HY-MiniSTM32V

Se pueden observar los 4 LEDS, situados uno en cada esquina de la placa y serigrafiados de LD1 a LD4. La función de cada uno de los LEDS se comenta a continuación.

- LD1: Pertenece al USB. Su labor principal es la de indicar transferencia de datos a través del puerto USB.
- LD2: LED de power. Se enciende cada vez que la placa tenga una alimentación de 5 V presente en uno de los pines del conector USB.
- LD3: LED de uso general. Conectado a una salida del procesador como indicador de lo que se desee.

- LD4: LED de uso general. Misma función que el LD3.

A parte de los LEDS también se pueden observar en la parte inferior de la placa los 4 botones antes comentados.

- Key A: Pulsador de uso general. Pertenece al puerto A.
- Key B: Pulsador de uso general. Pertenece al puerto B.
- Boot: Abreviatura de "Boot Loader". Sirve para colocar a la placa en modo de "carga de programa".
- Reset: Genera un reset al microcontrolador.

En la Figura 27 también se pueden observar las líneas de E/S del microcontrolador. Es aquí donde se tendrá que conectar el hardware externo necesario dependiendo de la aplicación que le estemos dando a la tarjeta de desarrollo.

Sin el display LCD, se aprecia la tarjeta como aparece en la Figura 28. En esta nueva imagen de la placa, se puede distinguir el microcontrolador y otros componentes. Se muestra un conector JTAG (Join Test Action Group), utilizado como sistema de depuración de código.

Figura 28. Parte frontal sin pantalla LCD de la placa HY-MiniSTM32V

La tensión de alimentación del microcontrolador, así como de la mayoría de elementos del circuito es de 3,3 V, pero en las pruebas normales la obtendremos a partir de los 5 V del USB del ordenador. La reducción la efectúa un integrado estabilizador, que tiene la forma de un pequeño transistor, situado al lado del conector JTAG.

2.5 Prestaciones del STM32F103VCT6

En este apartado se comentarán las características de aquellos periféricos que se han utilizado del STM32F103VCT6 durante la realización del proyecto.

2.5.1 GPIO (General Purpose Input/Output)

El STM32F103VCT6 consta de 80 pines de entrada/salida bidireccionales. Los pines están organizados en cinco puertos, teniendo cada puerto unos 16 pines.

Estos puertos están nombrados del A hasta el E y todos son tolerantes a 5 V. Casi todos los pines son multifunción y pueden ser cambiados de GPIO a una entrada/salida de un periférico, como puede ser la USART, el I2C o el ADC.

Cada GPIO tiene registros de configuración de 32 bits cada uno. Se hace una combinación de dos registros para tener un registro de configuración de 64 bits. Entre estos 64 bits, cada pin tiene un campo de 4 bits que permite definir su función.

Los pines GPIO pueden ser programados por software como salida, como entrada o como un periférico con función alternativa. Todos los pines GPIO soportan ciertas cantidades de corriente, si supera dicha cantidad, el microprocesador se estropea.

2.5.2 ADC (Analog to Digital Converter)

El STM32F103VCT6 tiene integrado tres convertidores analógicos-digitales de 12 bits de resolución cada uno con velocidad de muestreo de hasta 1 MHz. Ofrece una multiplexación de 18 canales, dónde 16 de ellos pueden servir para medir señales externas.

Cuando el ADC está configurado, es posible programar individualmente el tiempo de conversión de cada canal, habiendo 8 posibilidades de tiempos de conversión comprendidos entre 1.5 ciclos hasta 239.5 ciclos del ADCCLK (ADC Clock).

La configuración básica del ADC se realiza mediante dos registros. Estos registros son el registro de control y el registro de estado. Es aquí donde se indica el tiempo de muestreo, si queremos secuencias de muestreo de único disparo o de ráfaga, el canal en el que se quiere almacenar el dato, entre otras cosas. A parte de estos

registros se tiene después el registro de dato, que es donde se almacena el valor conertido por el ADC [26].

En este proyecto se ha utilizado el ADC1 del STMF103VCT6 con el fin de muestrear la señal que entra por uno de sus canales. La señal a muestrear por el ADC es la que se recibe de la tarjeta Muscle Sensor v3. Comentar que los valores almacenados por el ADC oscilan entre los 0 y 3.3 V.

2.5.3 USART (Universal Synchronous Asynchonous Receiver Transmitter)

El STM32F103VCT6 posee hasta 5 USART, cada uno de ellos con modos de operación que soportan las últimas aplicaciones en comunicaciones seriales. Todas las USART son capaces de trabajar con comunicaciones a velocidad de 4.5 Mbps. Cada USART es programable indicando la longitud del dato (8 o 9 bits), el bit de paridad, el bit de stop y la velocidad de transmisión.

Además la USART es capaz de una comunicación half-dúplex por único cable, utilizando únicamente el pin de transmisión (Tx). Para comunicaciones más modernas y flujo de control por hardware cada USART dispone de las líneas de control CTS (Clear To Send) y RTS (Ready To Send)[26].

En este proyecto se ha utilizado concretamente la USART1 para realizar las comunicaciones de los datos muestreados y procesados con la aplicación Android, donde también se visualizaría la señal que se representa en el TFT.

2.6 FTDI Chip DS-UMFT311EV

2.6.1 Introducción

El FTDI Chip DS-UMFT311EV [28] proporciona un puente entre la tarjeta de desarrollo HY-MiniSTM32V y el dispositivo Android en el cual se ejecutará la aplicación de monitorización de señales bioeléctricas. Permite a periféricos externos comunicarse con una plataforma Android ya que soporta interfaces tipo GPIO, UART, PWM, I2C Maestro, SPI Maestro y SPI Esclavo. El FTDI Chip se comunica con

dispositivos Android mediante un puerto USB. Es decir, es una especie de puente para las interfaces nombradas anteriormente y la interfaz USB.

Para conectarse con un dispositivo Android, ya sea smartphone, tablet o cualquier otro dispositivo, existen dos posibilidades: USB OTG (On The Go) y Open Accesory Mode [29]. El USB OTG es una extensión de USB 2.0 que permite poder conectar cualquier dispositivo mediante USB a nuestro smartphone o tablet. Esto se basa en que el dispositivo conectado pasa de ser un mero esclavo, a ser host. Lo que permite que se pueda acceder como maestro a los aparatos. Pero existe un problema, y es que no todos los dispositivos Android vienen con soporte de USB OTG, lo que significaría que no se podría utilizar esta forma de conexión entre dispositivos. Como no existe una lista oficial de cuáles dispositivos tienen USB OTG de fábrica o no, se ha querido evitar este tipo de conexión para la realización del proyecto.

Es por ello que se ha seleccionado el DS-UMFT311EV ya que implementa el modo Anroid Open Accesory. El modo Android Open Accesory tiene la misma finalidad que el USB OTG, la de convertir al periférico conectado al dispositivo en maestro y al soporte Android en esclavo, con la obligación de ser el periférico el encargado de alimentar el puerto USB a 5V. Para conectarse en este modo a cualquier equipo Android, este debe soportar dicho modo. Las plataformas Android que soportan el modo Android Open Accesory son aquellas versiones del 3.1 hacia delante, aunque también existen dispositivos con la versión 2.3.4 que lo soportan.

2.6.2 Características

La tarjeta DS-UMFT311EV [30] está basada en un único chip USB Android Host FT311D IC, con seis tipos de interfaces de comunicación seleccionables manualmente a través de 3 jumpers, las cuales son:

- 7 líneas de GPIO
- Interfaz UART básica
- 4 canales de PWM
- I2C Maestro

- SPI Esclavo soportando modos 0, 1, 2 y 3 y con opción de elección MSB (Most Significant Bit) o LSB (Less Significant Bit)

- SPI Maestro soportando modos 0, 1, 2 y 3 y con opción de elección MSB (Most Significant Bit) o LSB (Less Significant Bit)

A continuación se mostrará en la Figura 29 el FTDI Chip DS-UMFT311EV. En la figura se puede observar los distintos componentes y conectores que lo forman.

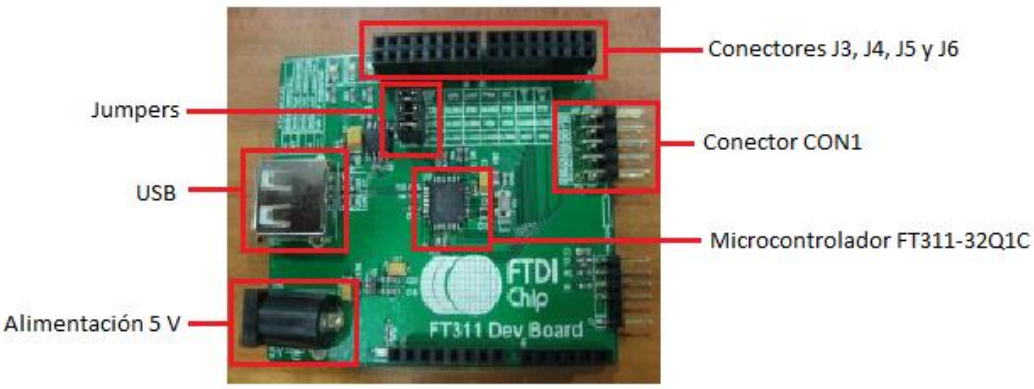

Figura 29. FTDI Chip DS-UMFT311EV y sus componentes

El conector CON1, según la configuración seleccionada, se utiliza como puerto de comunicación. Los conectores J3, J4, J5 y J6, que como el CON1, según la configuración seleccionada, se utilizan como puertos de comunicación de un tipo u otro. Los jumpers son para seleccionar la interfaz de comunicación deseada.

A continuación se describen las funciones de los componentes del diagrama de bloques del módulo FT311D. Módulos que se pueden apreciar en la Figura 30.

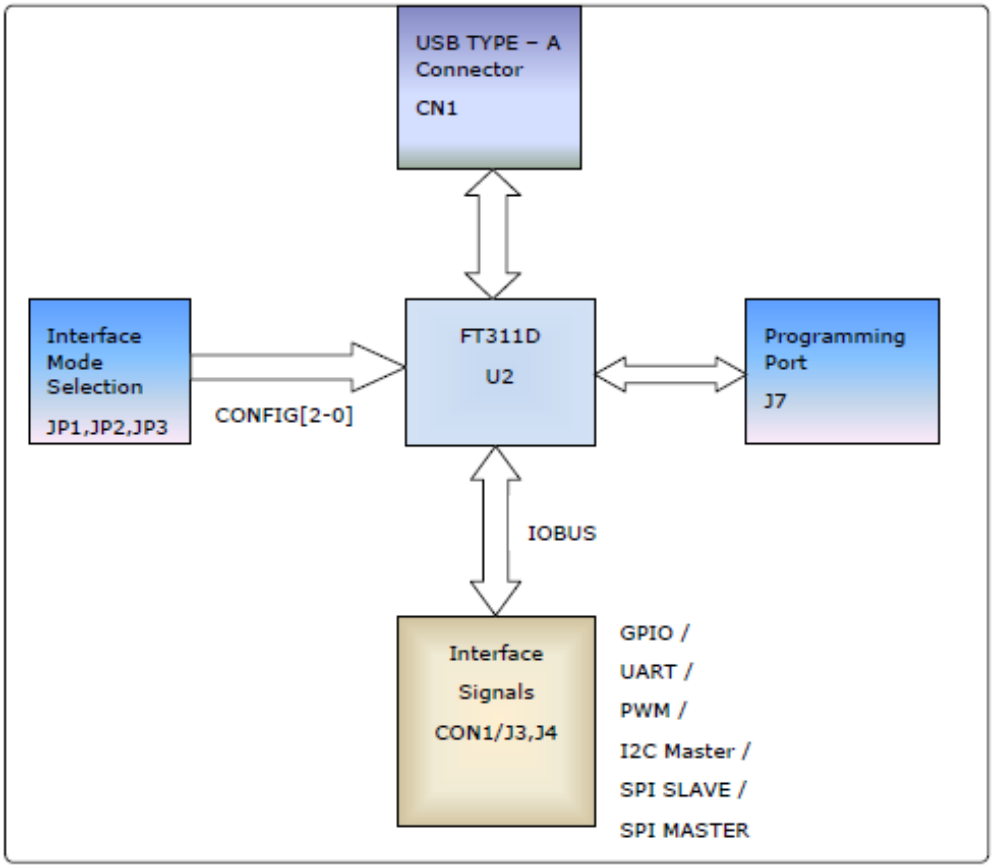

Figura 30. Diagrama de bloques del módulo FT311D

Modo de interfaz de selección (Interface Mode Selection): esto permite seleccionar manualmente el tipo de interfaz de comunicación que deseemos a través de los jumpers JP1, JP2 y JP3. Las configuraciones posibles haciendo uso de los jumpers son las siguientes:

Tabla 2. Configuración para selección modo de interfaz

Modo Interfaz	Selección interfaz		
	JP1	**JP2**	**JP3**
GPIO	0	0	0
UART	0	0	1
PWM	0	1	0
I2C Maestro	0	1	1
SPI Esclavo	1	0	0
SPI Maestro	1	0	1

USB Host: puerto USB Host en el conector CON1 que se utiliza para conectar el dispositivo Android que soporte el modo Android Open Accesory.

Señales de interfaz (Interface Signals): las señales en el CON1 dependerán del modo de interfaz seleccionado previamente.

Puerto de programación (Programming Port): el puerto de programación J7 se utiliza para reprogramar el dispositivo FT311D con un nuevo archivo ROM. El dispositivo no se suele reprogramar ya que viene listo para usar.

En la realización del proyecto, se ha seleccionado el modo de interfaz de la UART, ya que es por este medio por el que transmitimos los datos muestreados y almacenados en la tarjeta HY-MiniSTM32V hacia el soporte Android, pasando previamente por el DS-UMFT311EV. Esto quiere decir que hemos tenido que realizar la combinación de los jumpers (JP1, JP2, JP3) según descrito en la Tabla 2.

La interfaz UART en el módulo FT311D ofrece velocidades de transmisión desde los 300 hasta los 921600 bps. Los datos transmitidos por UART son en formato NRZ (Non Return to Zero). Las señales que se han usado en este módulo están disponibles en el CON1 y el J3, como se muestra en la Tabla 3.

Tabla 3. Señales UART módulo FT311D

Nombre señal	Conector CON1	Conector J3	Número pin	Tipo entrada/salida	Descripción
UART_TXD	CON1-10	J3-3	23	Salida	Transmisión datos
UART_RXD	CON1-9	J3-4	24	Entrada	Recepción datos

3. Desarrollo del sistema

3.1 Integración del sistema

3.1.1 Introducción

En este apartado se describen como se han realizado las distintas conexiones entre los equipos hardware utilizados para formar el sistema electrónico del trabajo.

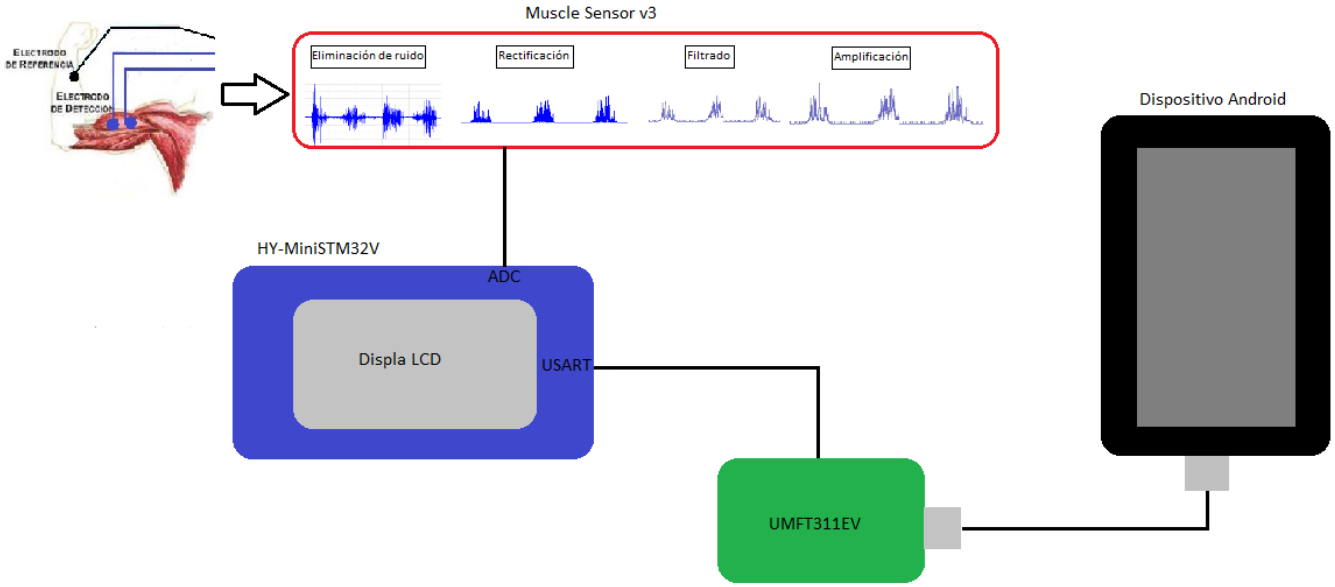

Figura 31. Integración del sistema final

A continuación se describe la integración de la etapa de la adquisición de la señal.

3.1.2 Etapa de adquisición de señal

Los elementos usados son el Muscle Sensor v3, la placa HY-MiniSTM32V y el ADC del STM32F103VCT6. A parte se hace uso de cables conductores para las distintas conexiones, una protoboard y componentes eléctricos (resistencia y diodos) para la realización de un circuito analógico.

La etapa de adquisición de señal es aquella en la que a través de la tarjeta de acondicionamiento de señal, y sus respectivos componentes, electrodos y cables con conector Jack, se obtiene la señal bioeléctrica.

Una vez llega la señal a la tarjeta de acondicionamiento, pasa por sus distintos circuitos analógicos, explicados con detalle en el capítulo 2 dedicado al Muscle Sensor v3, produciendo así a la salida una señal legible.

La salida del Muscle Sensor v3, serigrafiada en la placa como "*SIG*" (Véase Figura 20), se conecta a través de un cable conductor a la entrada del ADC del microprocesador a través de la placa HY-MiniSTM32V, concretamente en el pin 16 serigrafiado como "*PC1*".

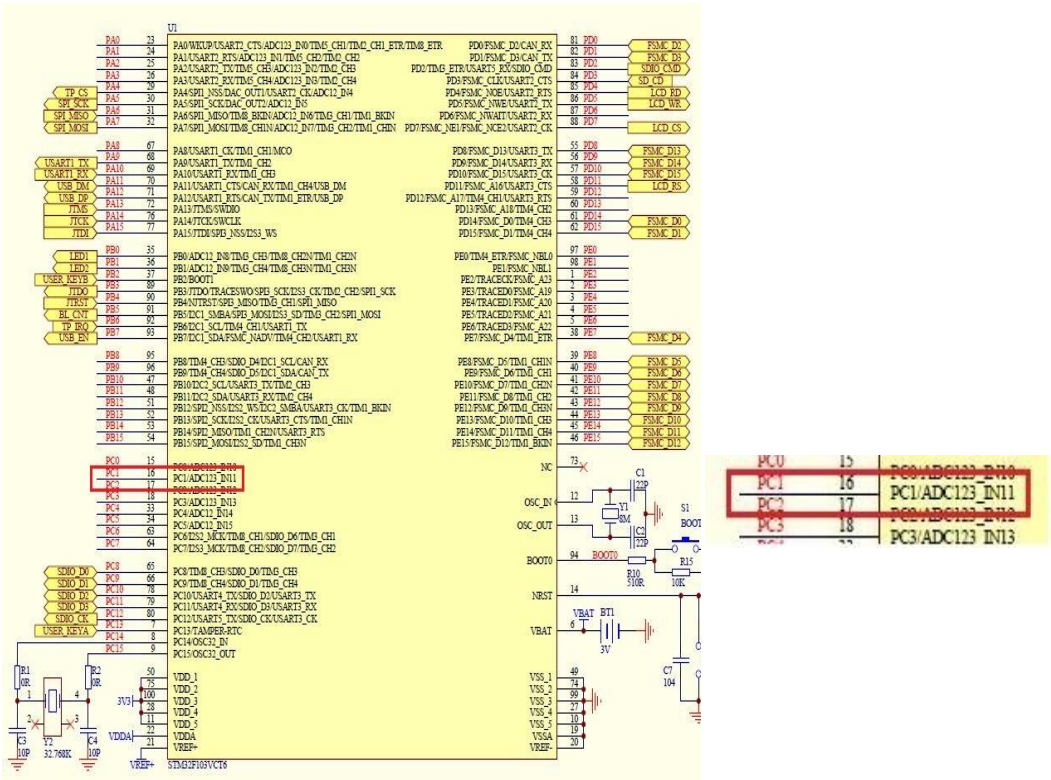

Figura 32. Pin ADC en placa HY-MiniSTM32V

Como se ha comentado, la señal del ADC fluctúa entre los valores 0 y 3.3 V, es decir en caso de recibir más voltaje por esa entrada se puede estropear el microprocesador. Ahora bien, también se ha detallado que la señal de salida del Muscle

Sensor v3 puede variar desde los 0 hasta la tensión de alimentación positiva (+Vs). En nuestro caso, se alimenta el sensor con ±5 V con la fuente de alimentación Tektronix CPS250, pudiendo alcanzar la señal de salida entonces dicho voltaje. Para evitar que a la entrada del ADC del microprocesador se alcancen estos valores se ha montado un circuito de protección (diodos clamping) en una protoboard por el cual pasa la señal antes de llegar a la placa de desarrollo. El circuito montado es el que se puede observar en la Figura 33.

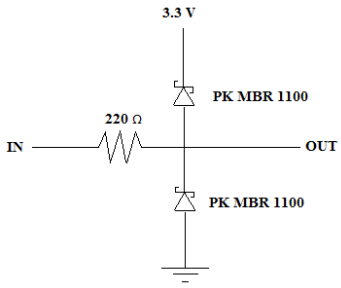

Figura 33. Montaje circuito de protección

Los 3.3 V que alimentan este circuito se aprovechan del HY-MiniSTM32V a través de su pin 48, que proporciona una tensión de 3.3 V.

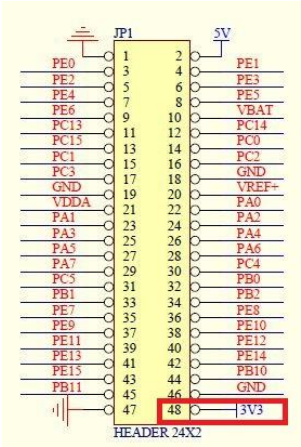

Figura 34. Pin 48 en placa HY-MINISTM32V

La etiqueta "*OUT*" va conectada a través de un cable conductor a la tarjeta de desarrollo al pin PC1, que es la entrada al ADC del microprocesador. El pin 46 "*GND*", es el que se utiliza como referencia para conectar el resto de las tierras. El conexionado de las tierras comunes se realiza en la protoboard.

3.1.3 Comunicación con plataforma Android

En este apartado se describe la integración entre los componentes necesarios para conseguir monitorizar la señal bioeléctrica en un dispositivo Android. Los dispositivos necesarios para esta conexión son la placa HY-MiniSTM32V, el FTDI Chip DS-UMFT311EV y la propia plataforma Android.

Esta etapa se realiza para mostrar en la pantalla de un dispositivo Android la señal bioeléctrica. Es decir, previo a esta comunicación, la señal ya ha sido procesada.

Para enviar los datos de la señal hacia el DS-UMFT311EV se ha optado para la comunicación serial USART. En concreto se ha utilizado la USART1 del STM32F103VCT6. Como únicamente se van a enviar datos, es decir, solo se va a utilizar la función de transmisión de la USART, únicamente hace falta configurar el pin de transmisión de la USART1. El pin correspondiente a la transmisión de la USART1 es el pin 68, serigrafiado como "*PA9*".

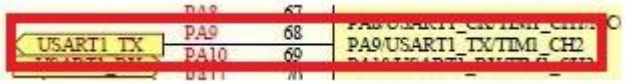

Figura 35. Pin USART1 TX en placa HY-MiniSTM32V

Una vez conectados los dispositivos anteriores, se ha de comunicar el DS-UMFT311EV con la plataforma Android. Para conseguir esta comunicación se requiere un cable USB/Micro-USB, ya que a través del puerto USB se transmiten las señales hacia el dispositivo Android. El USB irá al conector USB del DS-UMFT311EV y el Micro-USB irá conectado al puerto Micro-USB del soporte Android.

Figura 36. Cable USB/Micro-USB

En la Figura 37 la conexión de estos equipos a modo de diagrama de bloques.

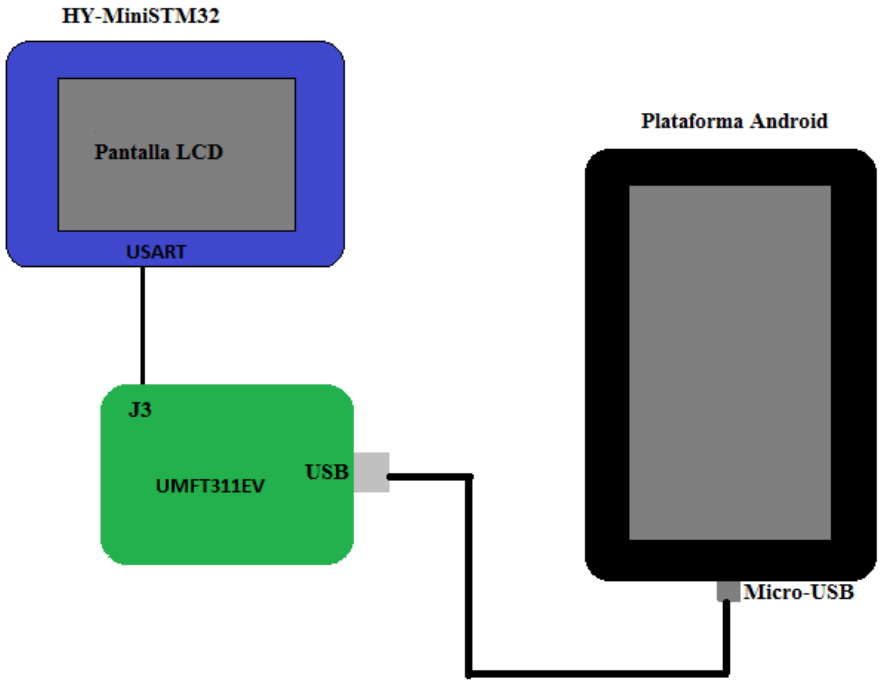

Figura 37. Diagrama de bloques de integración para comunicación con dispositivo Android

3.1.4 Prototipo de laboratorio

En la Figura 38 se podrá observar el montaje físico real de los dispositivos para su correcto funcionamiento.

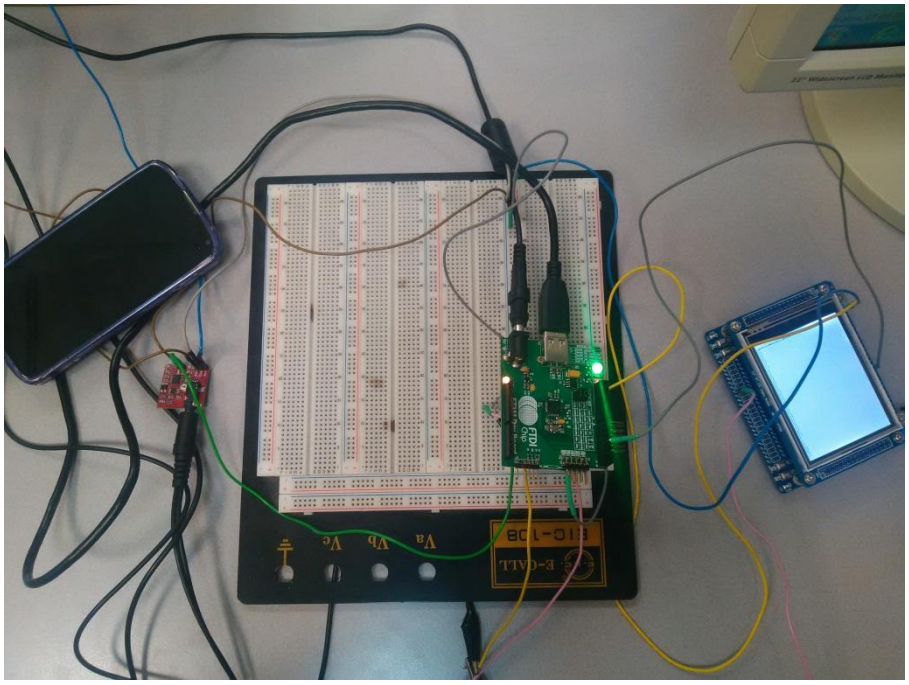

Figura 38. Montaje físico de la integración de equipos

3.2 Desarrollo Firmware

3.2.1 Introducción

En este apartado se va a explicar el código que ha sido desarrollado. El lenguaje de programación por el que se ha optado para el desarrollo del mismo ha sido el lenguaje de programación C.

3.2.2 Entorno de programación

Se ha estado diferenciado dos partes fundamentales en este proyecto. La adquisición de la señal y la representación de la misma por la pantalla LCD que provee la placa HY-MiniSTM32V, y la otra parte es la de la comunicación con un dispositivo Android para visualizar en el mismo la misma señal en base a una aplicación creada. Los entornos de programación utilizados han sido el Keil µVision 4.73 y el Android Studio.

- Keil µVision 4.73: es una plataforma de desarrollo, basada en Windows, que combina un editor y un gestor de proyecto. Incluye todo tipo de facilidades para desarrollar aplicaciones empotradas incluyendo un compilador C/C++ o assembler, un linker, un gestor de librerías y un conversor HEX. µVision compila, ensambla y enlaza la aplicación automáticamente y proporciona un único punto para focalizar los esfuerzos de desarrollo [31]. En la Figura 39 se puede apreciar un ejemplo de cómo es el entorno de programación.

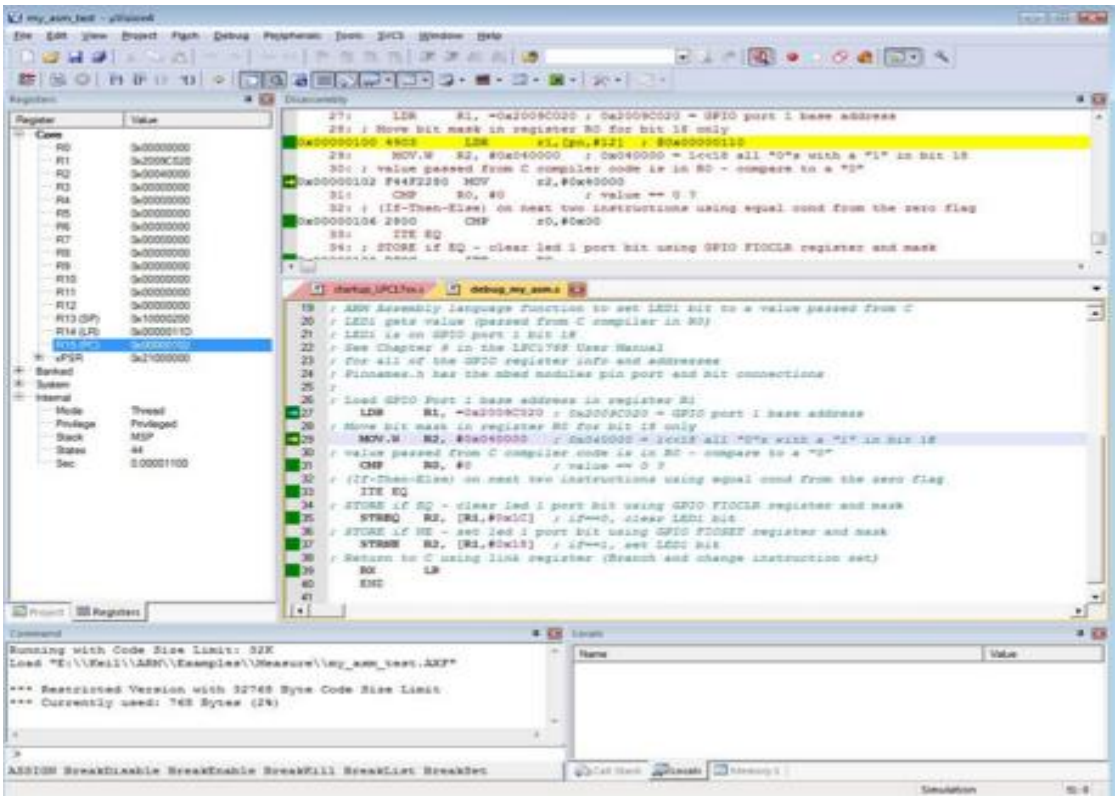

Figura 39. Entorno de programación Keil µVision 4.73

- Android Studio: Android Studio es un programa lanzado por Google y se trata del nuevo entorno de desarrollo integrado (IDE) para el sistema operativo Android y está basado en el software IntelliJ IDEA de JetBrains. Este entorno ofrece un editor de diseño que permite arrastrar y soltar los componentes en la interfaz de usuario, presentaciones de vista previa de múltiples configuraciones de pantalla, asistentes basados en un plantilla para crear diseños y componentes comunes Android, así como una serie de herramientas para detectar rendimiento, facilidad de uso, compatibilidad de versiones y otros problemas [32]. Se ha optado por este entorno de programación debido a que tiene una intuitiva interfaz de usuario y es totalmente gratuito. A continuación se observa en la Figura 39 el aspecto del mismo.

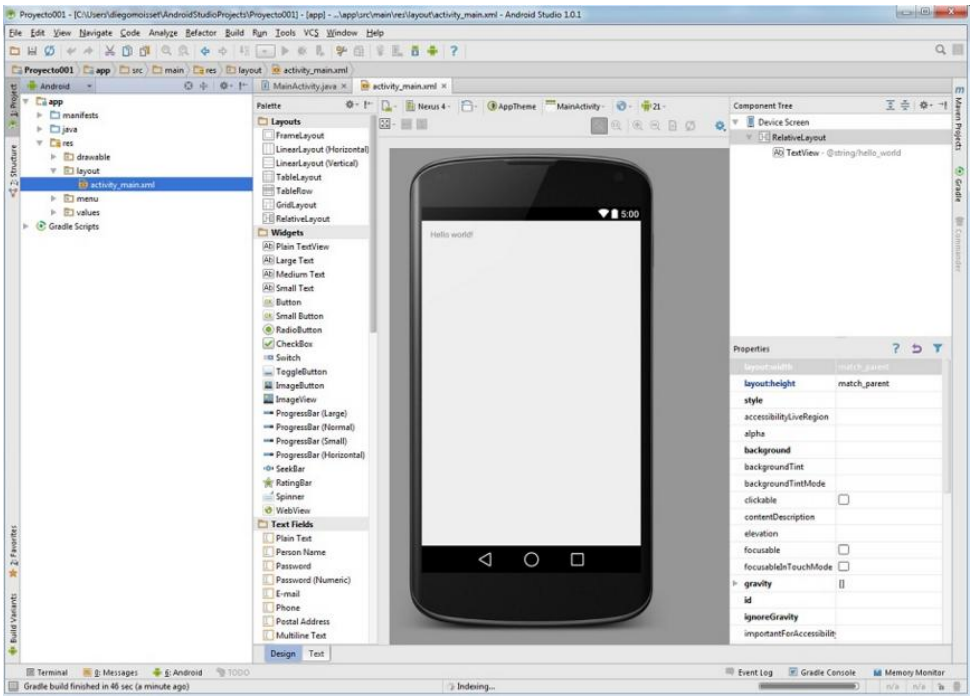

Figura 40. Entorno de programación Android Studio

3.2.3 Estructura del código

Esta parte hace referencia a la estructura del código desarrollado para el microprocesador STM32F103VCT6 en el entorno de programación Keil µVision4.

Como se ha comentado en el apartado anterior, el Keil µVision4 presenta un gestor de proyecto, el cual utilizamos para estructurar el código. La estructura seguida en el desarrollo del proyecto es la que se puede apreciar en la Figura 41. Si observamos la imagen, se pueden diferenciar 5 ficheros. Estos ficheros están divididos según la función y la contribución que realizan sobre el proyecto. A continuación se describirán aquellos dos archivos que han sido modificados para el desarrollo del proyecto.

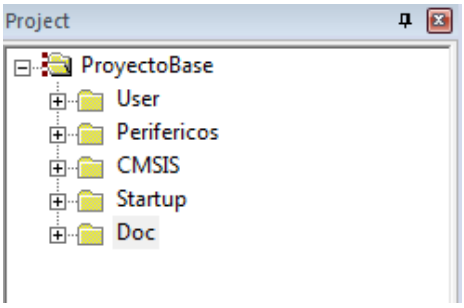

Figura 41. Estructura del código

- **User:** En este fichero se encuentran todos aquellos archivos principales en los cuales se programa correspondiendo a que se cumpla el objetivo de conseguir monitorizar la señal muscular deseada en la pantalla LCD del HY-MiniSTM32V.

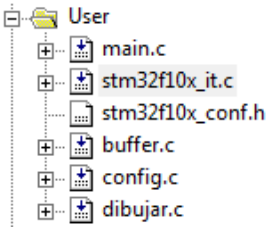

Figura 42. Archivos del fichero User

- **Perifericos:** En este fichero se encuentran todas aquellas librerías de los periféricos del sistema. Estas librerías fueron facilitadas por el fabricante. Los únicos archivos añadidos fueron el *GLCD.C y el TouchPanel.C*.

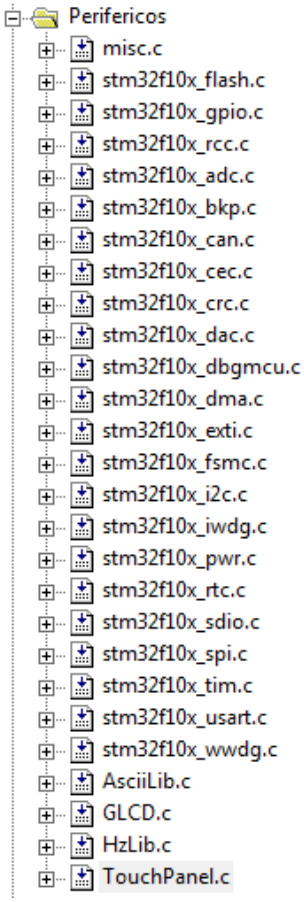

Figura 43. Archivos del fichero Perifericos

- **CMSIS:** Es una capa de abstracción hardware (HAL) que se encuentra desarrollada específicamente para la serie de procesadores ARM Cortex-M. Presentar esta abstracción permite la estandarización de las interfaces software de todos los dispositivos de este tipo por lo que permite la reutilización de software de una forma más simple.

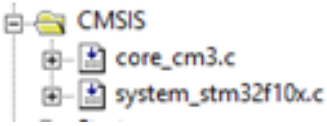

Figura 44. Archivos del fichero CMSIS

- **Startup:** En este fichero se incluyen los ficheros necesarios para la correcta inicialización del microprocesador.

Figura 45. Archivo del fichero Startup

- **Doc:** En este fichero se incluye un documento .txt cuya misión es hacer un resumen general del software realizado indicando sus principales funciones y características.

Figura 46. Archivo del fichero Doc

3.2.4 Código desarrollado

En este punto se explica el código que ha sido desarrollado a lo largo del proyecto para la monitorización de las señales biológicas musculares a través del TFT del HY-MiniSTM32V.

3.2.4.1 Configuración de periféricos y recursos

Hay un segmento de código que está enfocado a la configuración de aquellos periféricos y recursos que se utilizan del STM32F103VCT6. Los recursos y periféricos utilizados para la realización del proyecto son el ADC, la GPIO, el TIMER, la USART, entre otros.

En el fichero *main.c* es donde se realizan las inicializaciones de éstos, sin embargo es en la librería de periféricos donde se configura el uso que se quiere hacer del mismo. Estas inicializaciones es lo primero que se realiza en el programa principal. A continuación se presentarán segmentos de código para facilitar la explicación.

```
/* System clocks configuration ---------------------------------*/
  RCC_Configuration();

  /* NVIC configuration ------------------------------------------*/
  NVIC_Configuration();

  /* GPIO configuration ------------------------------------------*/
  GPIO_Configuration();

  /* TIM configuration -------------------------------------------*/
  TIM_Configuration();

  /* ADC configuration -------------------------------------------*/
  ADC_Configuration();

  /* TIM Enable --------------------------------------------------*/
  TIM_Enable();

  /* USART configuration -----------------------------------------*/
  USART_Configuration();

  /* GLCD configuration ------------------------------------------*/
  LCD_Initializtion();
  LCD_Clear(Blue);

  /* TouchPanel Inicializacion -----------------------------------*/
  //TP_Init();
  TouchPanel_Calibrate();
  config_visual();
```

Como se puede observar en el código anterior, todas las líneas están dedicadas a llamar a la configuración de los periféricos a utilizar.

- *RCC_Configuration():* se encarga de configurar los distintos relojes del sistema. Esta función ha sido utilizada para habilitar los relojes correspondientes al ADC1, al TIM2 y a la GPIOC.

- *NVIC_Configuration():* esta función ha sido utilizada para habilitar al TIM2 capaz de lanzar una interrupción cada vez que ocurra una evento. Cuando salta la interrupción la rutina de servicio (ISR) atiende a la función denominada *TIM2_IRQn.*

- *GPIO_Configuration():* esta función tiene como tarea la de indicar al STM32F103VCT6 que el pin 16 (PC0.1) es configurado como una entrada analógica, correspondiente al canal 11 del ADC1.

```
void GPIO_Configuration(void)
{
        GPIO_InitTypeDef GPIO_InitStructure;
        /* Configure PC.01 (ADC Channel11) as analog input */
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
        GPIO_Init(GPIOC, &GPIO_InitStructure);
}
```

- *TIM_Configuration():* la configuración del TIMER se ha realizado de tal manera que interrumpa al NVIC a una frecuencia de 833.33 Hz. Esta frecuencia es la que se ha decidido utilizar para la frecuencia de muestreo de la señal bioeléctrica.

```
void TIM_Configuration(void)
{
         TIM_TimeBaseInitTypeDef   TIM_TimeBaseStructure;
         TIM_OCInitTypeDef         TIM_OCInitStructure;
         /* TIM2 configuration -----------------------------------*/
        /* 72MHz/Prescaler(7200) = 10KHz ->
       10KHz/Frecuencia deseada = Period */
        /* Time Base configuration */
         TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);
         TIM_TimeBaseStructure.TIM_Period = 12 - 1;
        /* Muestreo a 833 Hz*/
        TIM_TimeBaseStructure.TIM_Prescaler = 7200 - 1;
         TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
         TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
         TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
```

```
        /* TIM2 channel1 configuration in Compare mode */
        TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Active;
        TIM_OCInitStructure.TIM_OutputState =  TIM_OutputState_Enable;
        TIM_OCInitStructure.TIM_Pulse = 1;
        TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
        TIM_OC2Init(TIM2, &TIM_OCInitStructure);
        TIM_ITConfig(TIM2, TIM_IT_CC2, ENABLE);
}
```

En el capítulo 1 se describió que el rango de frecuencias para señales bioeléctricas adquiridas mediante electrodos superficiales (SEMG) se reducía de 10 a 400 Hz, entonces:

$$F_{muestreo} > 2 \cdot F_{señal}$$

$$833.33\ Hz > 2 \cdot 400\ Hz \rightarrow 833.33\ Hz > 800\ Hz$$

El reloj del sistema (CLK) es de 72 MHz. Escogemos un valor para el *Prescaler* de 7200. Dividiendo el reloj del sistema entre el valor del *Prescaler* se consigue una frecuencia de 10 kHz. Entonces:

$$Periodo\ Timer = \frac{10\ kHz}{Frecuencia\ deseada} = \frac{10\ kHz}{800\ Hz} = 12.5 \rightarrow 12$$

Introduciendo entonces el valor calculado en la variable del periodo, se obtiene una frecuencia de muestreo tal que:

$$Frecuencia\ de\ muestreo = \frac{10\ kHz}{Periodo\ Timer} = \frac{10\ kHz}{12} = 833.33\ Hz$$

Ahora bien, para la señal bioeléctrica que proporciona el Muscle Sensor v3 a su salida Figura 47) no es necesaria una frecuencia de muestreo de estas magnitudes. Sin embargo, se ha optado por esta frecuencia para posibilitar el muestreo de la señal bioeléctrica antes de la etapa del filtrado (Figura 48) de la tarjeta de acondicionamiento.

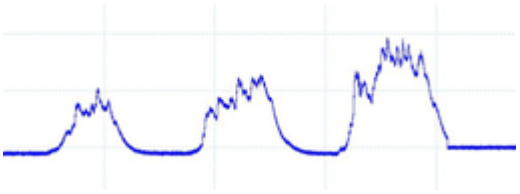

Figura 47. Señal EMG a la salida del Muscle Sensor v3

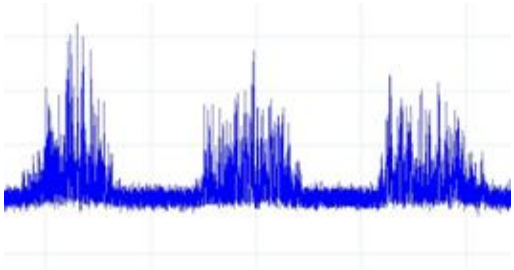

Figura 48. Señal EMG tras la etapa de rectificación

- *ADC_Configuration():* en esta función se configura el ADC1 de tal manera que escogemos el canal 11 para depositar los resultados del muestreo. Además se configura que el modo de muestreo sea a través de disparo único. Esto quiere decir que cada vez que se vaya a muestrear, se guarda en el canal 11 del ADC1 el valor exacto del momento de muestreo.

```
void ADC_Configuration(void){
	ADC_InitTypeDef ADC_InitStructure;
	/* ADC1 configuration ------------------------------------*/
	ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
	ADC_InitStructure.ADC_ScanConvMode = DISABLE;
	ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
	/* ADC_InitStructure.ADC_ExternalTrigConv =
	ADC_ExternalTrigConv_T2_CC2;  MODIFICADO */
	ADC_InitStructure.ADC_ExternalTrigConv =
	ADC_ExternalTrigConv_None; //MODIFICADO
	ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
	ADC_InitStructure.ADC_NbrOfChannel = 1;
	ADC_Init(ADC1, &ADC_InitStructure);
	/* ADC1 regular channel11 configuration */
	ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 1,
	ADC_SampleTime_1Cycles5);
	/* Enable ADC1 external trigger */
	ADC_ExternalTrigConvCmd(ADC1, DISABLE); //MODIFICADO
	/* Enable EOC interrupt */
	ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);
	/* Enable ADC1 */
	ADC_Cmd(ADC1, ENABLE);
	/* Enable ADC1 reset calibration register */
	ADC_ResetCalibration(ADC1);
	/* Check the end of ADC1 reset calibration register */
	while(ADC_GetResetCalibrationStatus(ADC1));
	/* Start ADC1 calibration */
	ADC_StartCalibration(ADC1);
	/* Check the end of ADC1 calibration */
	while(ADC_GetCalibrationStatus(ADC1));
}
```

- *USART_Configuration():* la USART2 ha sido configurada para enviar los datos a través del puerto serie. Los datos que se envían son los que se muestrean, y son enviados hacia el DS-UMFT311EV para que éste los envíe vía USB hacia el soporte Android. La transimisión se realiza a una velocidad de transmisión de 9600 bps, la longitud de las tramas es de 8 bits, con un bit de STOP y ningún bit de PARIDAD. También se realiza dentro de esta función la configuración GPIO necesaria para la transmisión de datos.

```
void USART_Configuration(void)
{
        GPIO_InitTypeDef GPIO_InitStructure;
        USART_InitTypeDef USART_InitStructure;
        RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOA |
        RCC_APB2Periph_USART1,ENABLE);
        /*  USART1_TX -> PA9 , USART1_RX ->         PA10 */
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOA, &GPIO_InitStructure);
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
        GPIO_Init(GPIOA, &GPIO_InitStructure);
        USART_InitStructure.USART_BaudRate = 9600;
        /* Velocidad de transmisión de 9600 bps */
        USART_InitStructure.USART_WordLength = USART_WordLength_8b;
        /* Longitud de trama = 8 bits */
        USART_InitStructure.USART_StopBits = USART_StopBits_1;
        /* Un bit de STOP */
        USART_InitStructure.USART_Parity = USART_Parity_No;
        /* Sin bit de PARIDAD */
        USART_InitStructure.USART_HardwareFlowControl =
        USART_HardwareFlowControl_None;
        USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
        USART_Init(USART1, &USART_InitStructure);
        USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
        USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
        USART_Cmd(USART1, ENABLE);
}
```

- *TouchPanel_Calibrate():* esta función es la rutina de calibración del Touch Panel (Pantalla LCD Resistiva). Con esta función se consigue calibrar la pantalla mediante la colocación de tres cruces, sobre los que hay que presionar, en la pantalla resistiva. Las dos primeras cruces se colocan en las

esquinas superiores del display, y la última cruz se coloca centrada en parte inferior del LCD. La primera de las cruces se puede observar en la Figura 49 Esta función ha sido creada para poder identificar el espacio temporal seleccionado en el menú programado en la función *config_visual().*

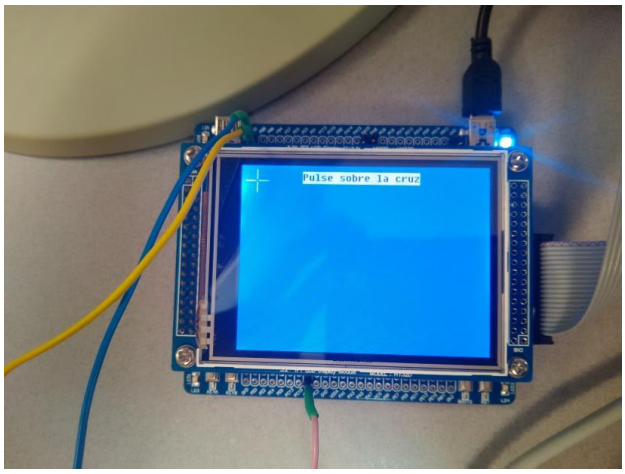

Figura 49. Calibración de la pantalla resisitiva

- *config_visual():* con esta función lo que se consigue es dar al usuario una serie de opciones de tiempo en las que quiera que se represente la señal. Estos espacios temporales son 6, 3, 2 o 1 segundo. El tiempo elegido tiene repercusión sobre la función que se encargue de dibujar la señal sobre la pantalla LCD. Se muestra en la Figura 50 el menú d selección de tiempo.

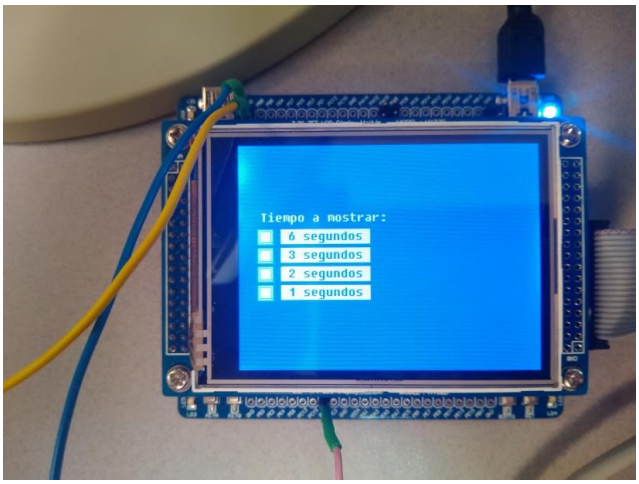

Figura 50. Menú de selección de tiempo para la representación de la señal bioeléctrica

3.2.4.2 Adquisición de la muestra y representación de la misma

Las variables que se han creado en el programa principal son las que se encuentran a continuación.

```
int main(void){
        static int i = 0;
        static int j = 0;
        static float cociente = 0;
        static float result;
        static uint16_t valor_pixel;
        static uint16_t dato;
        static uint16_t promedio = 0;
        static uint16_t aux = 0;
        uint8_t vuelta = 0;
        uint16_t x = 0;
        uint16_t datoAlto, datoBajo;
        uint8_t contador= 0;
        static uint8_t datoTx1, datoTx0;
```

Con la frecuencia de muestreo de trabajo, se realiza un disparo cada 0.0012 segundos (aproximadamente). Esto quiere decir que cada 0.0012 segundos la muestra que está almacenada será sobrescrita por una nueva muestra. Es por ello que se tienen que almacenar los valores de la muestra. La opción por la que se ha optado es la de crear una cola circular de valor 100. Esta cola circular será alimentada por las muestras que se vayan adquiriendo, y a la vez es la que proporciona los valores para dibujar la muestra en la pantalla.

Para eliminar ruido de cuantificación, lo que se hace es colocar el comando que indica al ADC1 que realice un disparo en un bucle *for*, en el que se dispare cuatro veces. Estos cuatro disparos tienen una diferencia ínfima de tiempo entre sí, por lo que los valores serán muy similares. Entonces, cada vez que se dispara, el valor muestreado nuevo se suma al anterior. Una vez realizados estos cuatros disparos, se divide el valor sumado de los disparos entre cuatro, obteniendo así un valor promedio de los valores muestreados. Y este valor promedio es el que proporcionamos la cola circular.

El ADC del STM32F103VCT6 tiene una resolución de 12 bits, por lo que el ADC puede dar hasta 2^{12} = 4096 valores distintos. Desde el valor 0 para 0 V hasta el

valor 4095 para el valor 3.3 V. Esto implica que hay que encontrar una relación entre los 233 píxeles en vertical y los 4096 valores que nos puede proporcionar el ADC. Para conseguir esta relación lo que se hace es multiplicar el valor muestreado por el valor de un coeficiente obtenido del cociente.

$$cociente = \frac{píxeles}{valores\ ADC} = \frac{233}{4096} = 0.05688$$

Multiplicando el cociente por el valor proporcionado por el ADC, conseguimos como resultado el valor del píxel correspondiente. Sin embargo, habrá aproximadamente unos 17 valores del ADC que se representarán en el mismo píxel.

$$\frac{píxel}{cociente} = \frac{1}{0.05688} = 17.5808$$

Esto quiere decir, que los primeros 17 valores del ADC (0-16) serán representados en el píxel 0. Los siguientes 17 valores (17-33) serán representados en el píxel 1, y así sucesivamente. Este efecto se ve minimizado en Android, ya que cualquier dispositivo tiene mayor resolución.

Para la representación de la señal en la pantalla LCD, se utilizan los valores del píxel correspondiente al último valor muestreado y justo el anterior, es por ello que no se puede reutilizar la cola circular y se debe crear un vector donde se puedan almacenar los valores. El vector que se crea es de tamaño 100. La técnica que se utiliza para dibujar es la de trazar una línea desde el valor anterior hacia el último.

Con los detalles explicados se consigue una representación aceptable en cuanto a la señal real.

Existe un espacio temporal para la representación de la señal, por lo que una vez completado el tiempo, la pantalla completa se encuentra con la señal dibujada. Es por ello que se hace uso de una ventana deslizante blanca que va barriendo la señal antigua para dejar sobre la pantalla la señal a tiempo real.

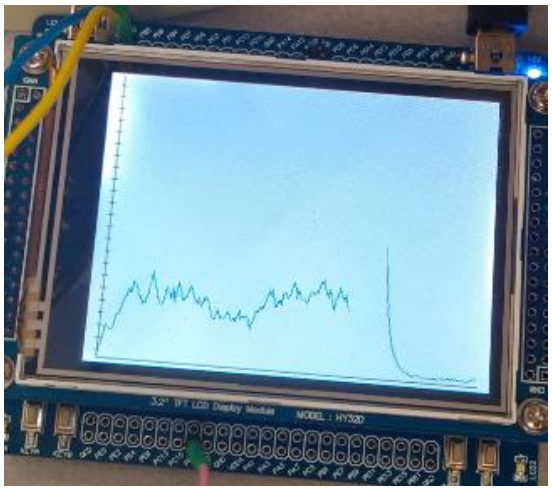

Figura 51. Proceso de limpieza de señal anterior

A continuación se muestra el segmento de código detallado anteriormente, y que representa la señal bioeléctrica sobre el TFT.

```
While (1){
        while(convertir == 0);   /*Esta es la variable que cambia de valor una vez se entre en
        la ISR del timer1*/
        for (j = 0; j< 4; j++){
                ADC_SoftwareStartConvCmd(ADC1, ENABLE);
                /*Habilitamos el disparo por Software del ADC*/
                while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC)==RESET);
                aux = ADC_GetConversionValue(ADC1) /*Se coge el valor proporcionado por
                el ADC*/
                promedio = aux + promedio; /*Se va sumando a la variable promedio el valor
                que se va adquiriendo en cada momento del ADC*/
        }
        ADC_ClearITPendingBit(ADC1, ADC_IT_EOC);
        promedio = (promedio/j); /*Se calcula el promedio del total de unas 4 muestras que se
        adquieren casi simultáneamente*/
        Ejes();
        push (&buffer2, promedio);//Se mete en la cola circular el dato promediado en la parte
        superior de este código*/
        resultado = popFast(&buffer2);
        dato = *resultado;
        result = (((float)dato)*(cociente)); /*Se calcula el pixel a pintar*/
        valor_pixel = (uint16_t) result; /*en esta variable se guarda el pixel a pintar*/
        if(valor_pixel>=233){
                valor_pixel = 233; /*Forzamos a que el valor máximo de esta variable sea 233
                para que en la resta no de menos de 2 en ningún momento */
        }
        valor_pixel = 235 - valor_pixel;
        almacen_Datos[i] = valor_pixel;
        actual = almacen_Datos[i];
        i++;
```

```
if (i > 1){
        if (vuelta > 0){
                x = eje_x;
                for (j = 0; j< 30; j++){
                        LCD_DrawLine(x, 2, x, 235, White);
                        x++;
                }
        }
        anterior = almacen_Datos[i-2];
        if(i == 100){
                almacen_Datos[0] = almacen_Datos[i-1];
                i = 1;
        }
        dibujar_Linea2(eje_x,actual, (eje_x-incremento_x), anterior, Blue);
        contador = 0;
}
eje_x = eje_x + incremento_x;
if (eje_x >= 310){
        eje_x = 10;
        vuelta = 1;
}
```

3.2.4.3 Transmisión de datos por USART

En este punto se presenta el código desarrollado para transmitir los datos, proporcionados por la cola circular, a través de la USART1 hacia el receptor de la USART en el DS-UMFT311EV. Para conseguir una exacta transmisión de los datos se ha tenido que crear un protocolo de comunicación incluyendo cabeceras en las tramas. Esto ha sido necesario porque la longitud de las tramas para enviar por la USART1 son de 8 bits, mientras que los valores del ADC se guardan en variables de 16 bits, aunque solo sean relevantes los primeros 12 bits. Entonces lo que se ha establecido es dividir los 12 bits relevantes en dos tramas de 8 bits, donde los primeros 2 bits son una cabecera, y los restantes 6 son los que corresponden al valor del ADC.

Tenemos una variable de 16 bits que se llama *dato* y las dos tramas a enviar por la USART son *datoAlto datoAlto* y *datoBajo* respectivamente.

Para la primera trama (*datoAlto)* se realiza una AND lógica con el valor en hexadecimal FC0 (111111000000 en binario) y el valor del ADC para conservar los 6 primero bits del resultado. A continuación se desplazan estos 6 bits a la parte menos significativa de la trama, para conservar los 2 bits más significativos para la cabecera.

La cabecera se introduce realizando una OR lógica con el valor hexadecimal 40 (01000000 en binario) quedando de esta manera la trama resultante con el valor binario de 01XXXXXX (X puede representar 1 y 0). A continuación se puede observar el resultado de las tramas de *datoAlto*, además del segmento de código dedicado a este protocolo.

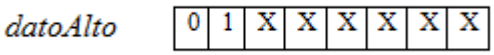

Figura 52. Trama correspondiente a los 6 bits más significativos del valor muestreado (datoAlto)

```
/* Se realizan las operaciones necesarias para coger los primeros 6 bits de datos del ADC*/
datoAlto = ((dato & 0xFC0) >> 6);
datoAlto = (datoAlto | 0x40);
datoTx1 = datoAlto;
```

Para la segunda trama se hace básicamente lo mismo atendiendo a que los bits que nos interesan ahora no son los 6 primeros bits, si no los siguientes 6, es por ello que el valor con el que se realiza la AND lógica varía. Para la cabecera de la segunda trama se ha establecido que se realice una OR lógica con el valor en hexadecimal 80 (10000000 en decimal), quedando una trama resultante con el valor binario 10XXXXXX. A continuación se muestra la trama y el código.

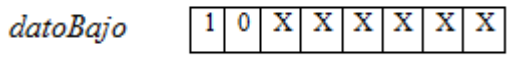

Figura 53. Trama correspondiente a los 6 bits menos significativos del valor muestreado (datoBajo)

```
/* Se realizan las operaciones necesarias para coger los ultimos 6 bits de datos del ADC*/
datoBajo = ((dato & 0x3F) | 0x80);
datoTx0 = datoBajo;
```

Por último no queda si no enviar estas dos tramas por la USART1 hacia el destino. Aquí el código empleado.

```
/* Enviamos por la USART1 los primeros 6 bits con cabecera 0x40*/
while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
        USART_SendData(USART1, datoTx1);
       /* Enviamos por la USART1 los ultimos 6 bits con cabecera de 0x80*/
while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET);
        USART_SendData(USART1, datoTx0);
```

3.3 Aplicación Android

Para desarrollar la aplicación Android han sido necesarios integrar los equipos HY-MiniSTM32V y el DS-UMFT311EV, que actúa como puente hacia el soporte Android. La explicación debida a la elección de este componente se encuentra en el capítulo 2 dedicada al FTDI Chip DS-UMFT311EV.

El entorno de programación utilizado para desarrollar esta aplicación es la de Android Studio, por lo que el lenguaje de programación utilizado ha sido Java.

Para configurar el DS-UMFT311EV en modo UART hay que hacerlo tanto por software como por hardware. La configuración hardware de este dispositivo es según una combinación de jumpers en los pines configuración para elección de comunicación. Los jumpers han tenido que ser colocados como se aprecia en la Figura 54.

Figura 54. Colocación jumpers para seleccionar comunicación por UART

La otra parte de la configuración se realiza por software, en donde se utiliza la librería del fabricante para indicar las configuraciones básicas de una UART, como puede ser la velocidad de transmisión, el número de bits, etc. A continuación se muestra el código de la configuración por Software.

```
public void init_UART() {
        baudRate = 9600; /* baud rate */
        stopBit = 1; /* 1:1stop bits, 2:2 stop bits */
        dataBit = 8;/* 8:8bit, 7: 7bit */
        parity = 0; /* 0: none, 1: odd, 2: even, 3: mark */
        flowControl = 0; /* 0:none, 1: flow control(CTS,RTS) */
        uartInterface = new FT311UARTInterface(this);
        uartInterface.SetConfig ( baudRate, stopBit, dataBit,    parity,   flowControl );
}
```

Una vez configurado el modo de comunicación se desarrolla el código para recibir las tramas del HY-MiniSTM32V. Es por ello que en el código desarrollado se identifican cada una de las cabeceras para volver a crear una única trama que coincida con el valor original muestreado. Una vez recreada la trama original, se utiliza para su representación en la pantalla del soporte Android. El código dedica a ello se muestra a continuación.

```
private double generateData() {
      double result = 0;
      int dataFirstFrame = dataReceived[read];
      int dataSecondFrame = dataReceived[read + 1];
      int firstFrameMask = 0x40;
      int secondFrameMask = 0x80;

      if ((dataFirstFrame & firstFrameMask) != firstFrameMask) {
            read++;
      } else if ((dataFirstFrame & firstFrameMask) == firstFrameMask) {
                  dataFirstFrame = ((dataFirstFrame & 63) << 6);
                  if ((dataSecondFrame & secondFrameMask) == secondFrameMask) {
                        dataSecondFrame = (dataSecondFrame & 63);
                  }
                 read += 2;
                 if (read == 4096) {
                        read = 0;
                 }
       result = (double) (dataFirstFrame | dataSecondFrame);
      }
      return result;
}
```

Tanto la parte de recepción de datos como la parte de representación de los mismos se han realizado a partir de las librerías Open Source creadas por Jonas Gehring [33].

4. Resultados

En este capítulo se presentarán los resultados obtenidos del proyecto desarrollado y junto a ello la comparación de objetivos.

El músculo por el que se ha optado para la prueba es el bíceps (Figura 55). Para distinguir señal muscular en los dispositivos de monitorización, el ejercicio que debe realizar el voluntario es el de una contracción voluntaria del músculo en cuestión.

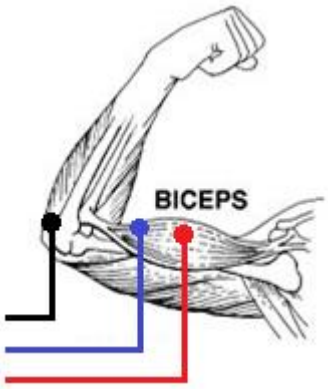

Figura 55. Músculo bíceps del cuerpo humano

En la Figura 56 se muestra la colocación de los electrodos sobre el músculo en cuestión.

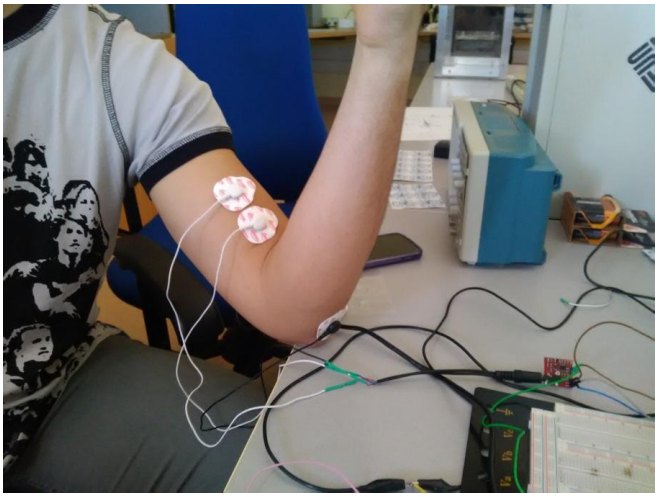

Figura 56. Colocación electrodos para músculo bíceps

Estos electrodos conducen la señal a la tarjeta de acondicionamiento Muscle Sensor v3 por la que la señal pasa por distintas etapas. Para visualizar el efecto que reproducen estas etapas sobre la señal se ha hecho uso de las sondas SP-260 60 MHz y el osciloscopio PicoScope 3224. Se han ido pinchando las sondas sobre el sensor en los diferentes circuitos para visualizar la señal en el osciloscopio. Donde primero se ha realizado la medición de la señal es en la etiqueta *MEASURE* que se encuentra a la salida del amplificador diferencial (Figura 57). Y en la Figura 58 el resultado de la señal en este punto del sensor. La representación de la señal en el osciloscopio tiene un espacio temporal de 9 segundos correspondientes a las 4 contracciones.

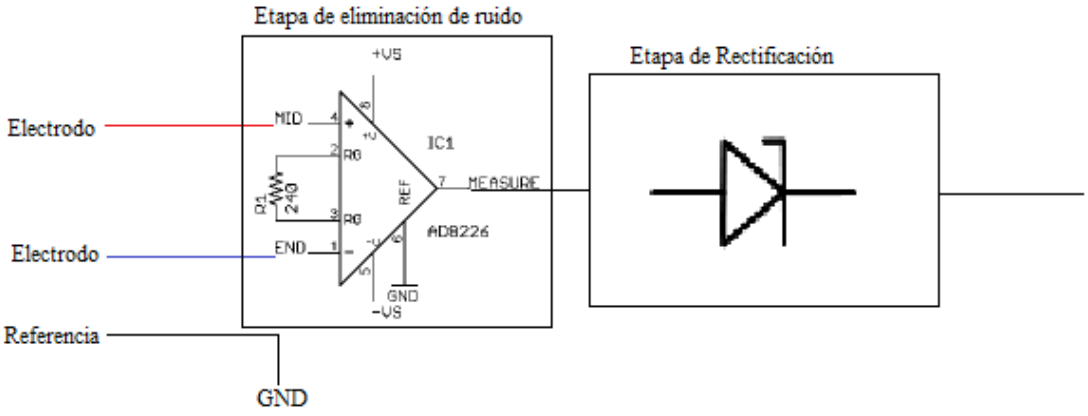

Figura 57. Etiqueta *MEASURE* en el esquemático del Muscle Sensor v3

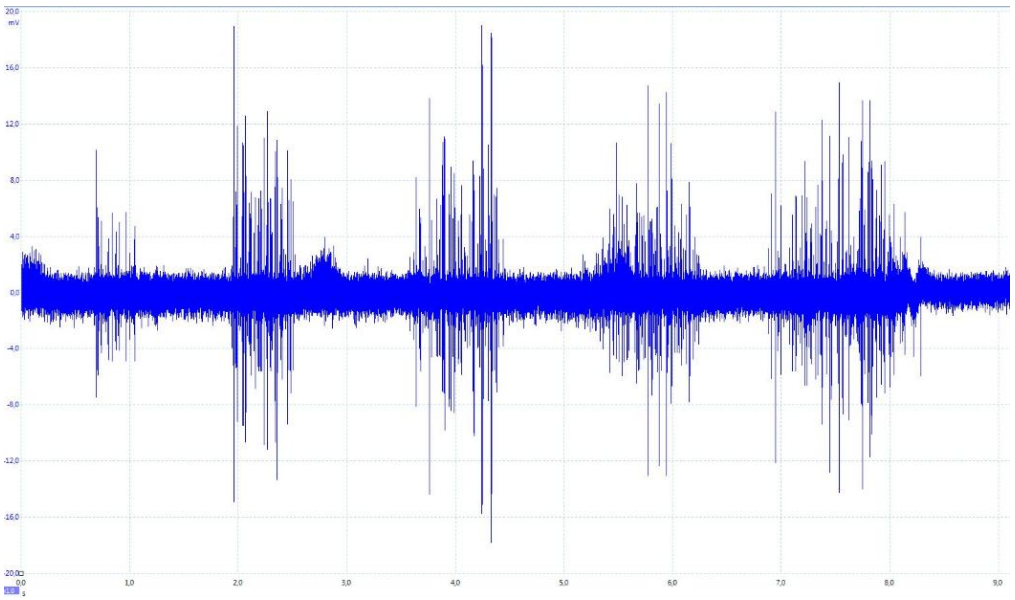

Figura 58. Señal a la salida del amplificador diferencial

Aquellos puntos en los que señal presenta valores es debido a que se ha realizado una contracción voluntaria del músculo. En este punto la señal aún presenta valores negativos, de ahí que se necesita una etapa de rectificación. Lo que se puede concluir debido a la Figura 58 es que el amplificador diferencial elimina las señales interferentes.

La siguiente etapa a la que se enfrenta la señal es a la de rectificación. En este, la medición se ha realizado en la etiqueta *RECTIFY*. En la Figura 59 y Figura 60 se puede apreciar, primero el punto al que corresponde en el esquemático y, segundo, el efecto que tiene sobre la señal este circuito.

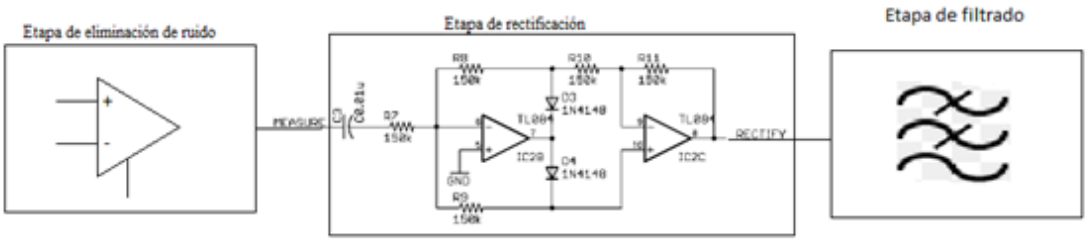

Figura 59. Etiqueta *RECTIFY* en el esquemático del Muscle Sensor v3

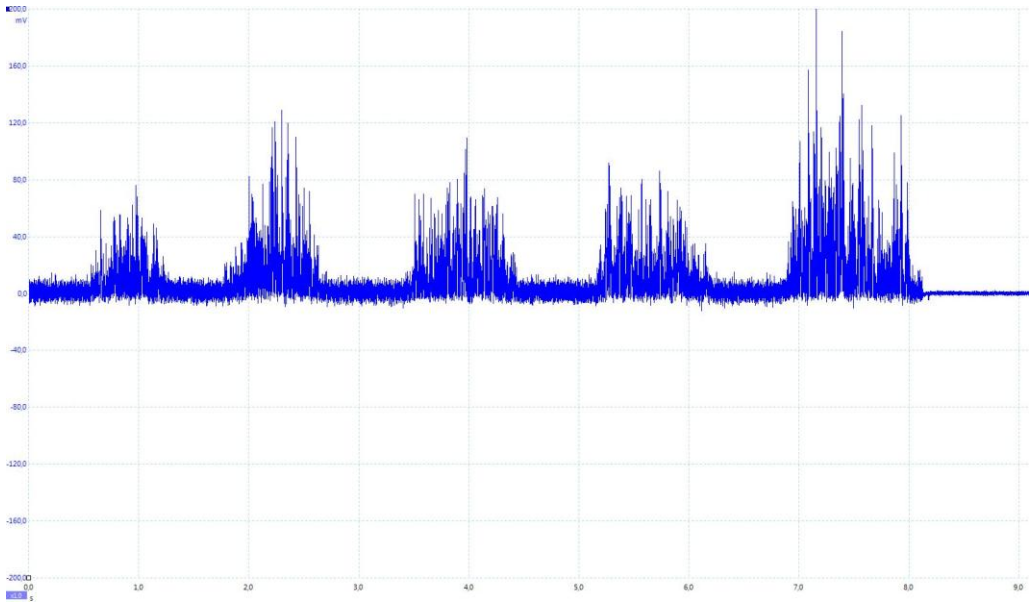

Figura 60. Señal a la salida de la etapa de rectificación

Las siguientes medidas sobre el Muscle sensor v3 se realizaron en la etiqueta *SIG*. De aquí la señal bioeléctrica es conducida a la entrada del ADC del STM32F103VCT6, el pin "*PC1*" del HY-MiniSTM32V.

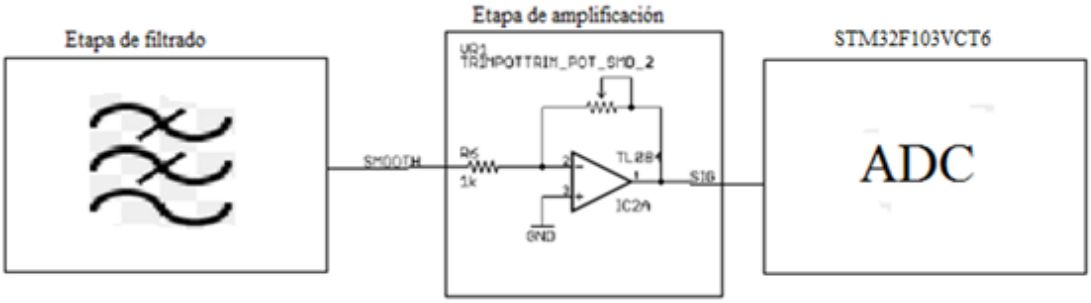

Figura 61. *SIG* en el esquemático del Muscle Sensor v3

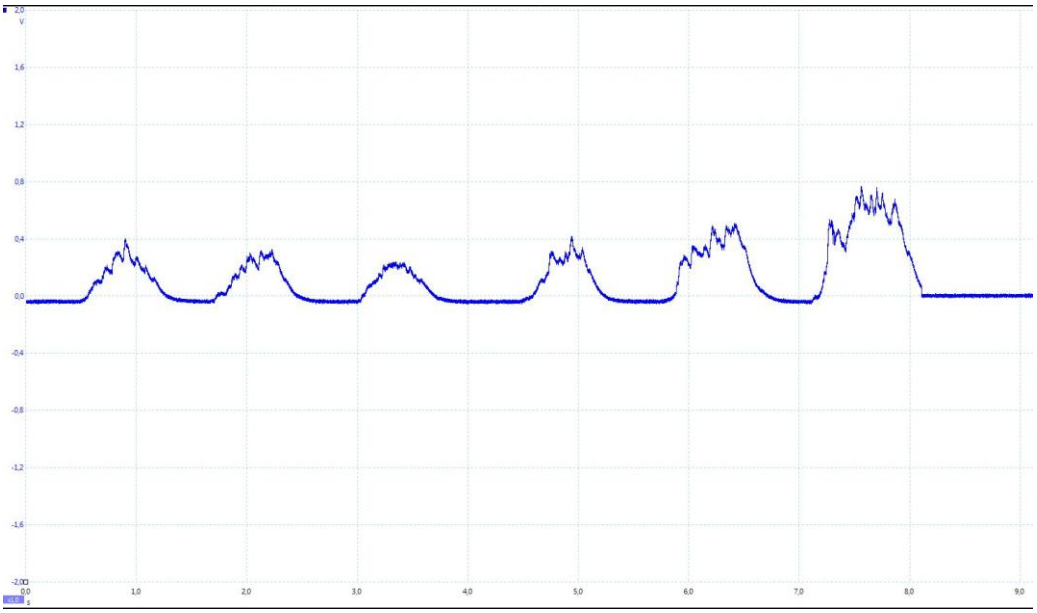

Figura 62. Señal a la salida del Muscle Sensor v3

Las señales bioeléctricas representadas por el display del HY-MiniSTM32V han sido comparadas con la señal real a la salida del Muscle Sensor v3, que ha sido medida a través de las sondas y el osciloscopio.

Para las pruebas realizadas, la selección temporal escogida en el programa del HY-MiniSTM32 fue de 6 segundos para facilitar la realización de fotos. En la primera medición se realizaron varias contracciones voluntarias, cuya señal representada en el osciloscopio se aprecia en la Figura 63. Esta señal es la que se compara con la de la Figura 64 y se puede apreciar como efectivamente, las señales son prácticamente iguales, pudiendo diferenciar y relacionar las contracciones realizadas.

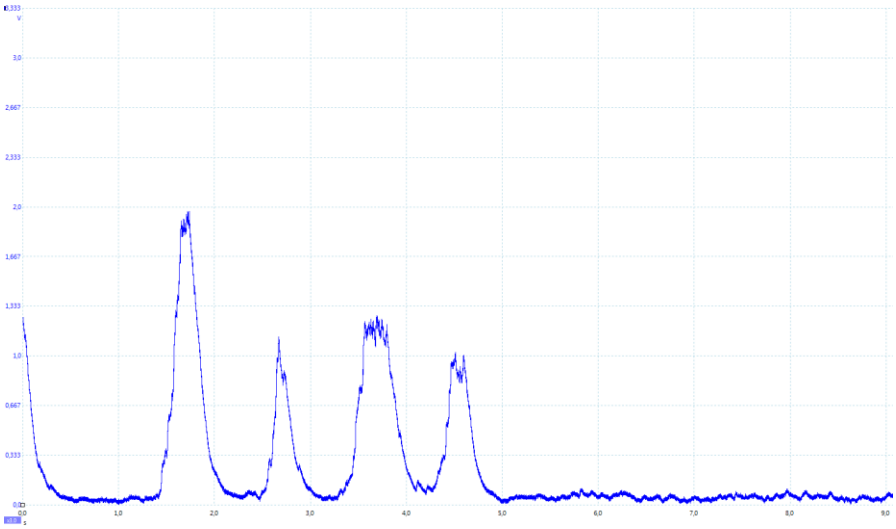

Figura 63. Señal de varias contracciones mostrada en el PicoScope 3224

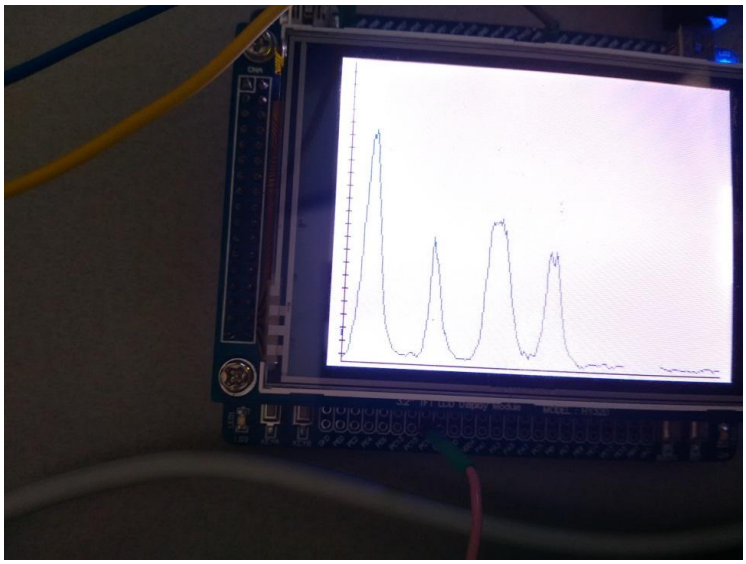

Figura 64. Señal de varias contracciones en la pantalla del HY-MiniSTM32V

También se presentan a continuación la monitorización de las señales musculares en el soporte Android gracias a la aplicación creada. Se contrastará la señal con su equivalente en la pantalla LCD del HY-MiniSTM32V

Figura 65. Representación señal muscular en soporte Android

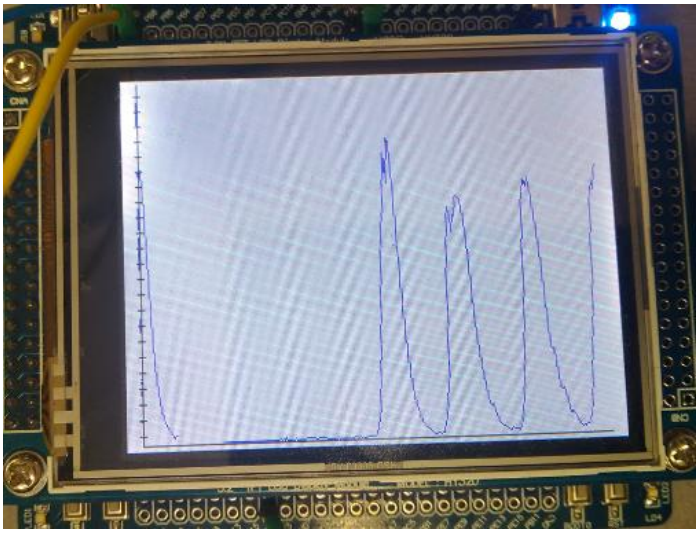

Figura 66. Representación misma señal Figura 59 por pantalla LCD

Una vez conseguido representar la señal muscular tanto por la pantalla LCD del HY-MiniSTM32V y por el soporte Android se recibe la señal en un principio por el ADC del STM32F103VCT6 (Figura 63) se determeina que los objetivos marcados en un principio han sido alcanzados.

5. Conclusiones y líneas futuras

El objetivo principal de este proyecto era el de monitorizar las señales biológicas musculares a tiempo real sobre un soporte Android como si de una EMG (Electromiografía) se tratase. Para ellos se ha hecho uso de recursos materiales electrónicos como recursos software. Todos los recursos software y muchos de los recursos materiales han sido aportados por la división de Equipos y Sistemas de Comunicación del IUMA.

Para lograr esta representación se ha hecho uso de la tarjeta de acondicionamiento de señal Muscle Sensor v3, que conectada a la tarjeta de desarrollo HY-MiniSTM32V cumplía la función de adquisición y representación de la señal muscular sobre la pantalla LCD de la tarjeta de desarrollo. Se ha desarrollado para ello un código en el lenguaje de programación C en el entorno de programación Keil µVision4.

Para conectar la placa HY-MiniSTM32V con el soporte Android se ha tenido que utilizar como puente el chip DS-UMFT311EV, debido a que la placa de desarrollo no puede enviar datos a través de ningún puerto USB.

Una vez lograda la comunicación entre el soporte Android y la placa HY-MiniSTM32V. Se ha creado una aplicación para representar la señal muscular en la pantalla del soporte Android utilizado. El lenguaje de programación utilizado para esta aplicación fue Java y el entorno de programación Android Studio.

Como línea futura existe la posibilidad de migrar el código desarrollado para una placa con dimensiones menores como es el Maple Mini r2, con exactamente el mismo microprocesador (STM32F103VCT6). La idea al migrarlo a esta placa sería la de crear un prototipo “wearable”, aunque esto implicaría desarrollar un protocolo de comunicación inalámbrica y la inclusión de baterías para el mantenimiento de la placa.

Otra posibilidad que ofrece la el proyecto desarrollado es el de controlar a deportistas profesionales para intentar crear un patrón de fatiga muscular y de esta manera evitar lesiones.

6. Presupuesto

El presupuesto de este Trabajo de Fin de Grado que se va a detallar a continuación es orientativo siguiendo las recomendaciones del Colegio oficial de Ingenieros Técnicos de Telecomunicación (COITT) que ofrece una herramienta orientativa para trabajos profesionales, el presupuesto se ha desglosado en varias secciones indicando los distintos costes orientativos asociados al desarrollo del proyecto. Estos costes se dividen en:

- Recursos materiales.
- Trabajo tarifado por tiempo empleado.
- Costes de redacción del Trabajo Fin de Grado.
- Material fungible.
- Derechos de visado del COITT.
- Costes de tramitación y envío.
- Aplicación de impuestos.

6.1 Recursos materiales

Para la realización de este proyecto han sido necesarios tanto recursos hardware como recursos software.

6.1.1 Recursos hardware

Los recursos materiales utilizados a lo largo del desarrollo del TFG son los siguientes:

- Tarjeta de acondicionamiento de señal Muscle Sensor v3
- Placa de desarrollo HY-MiniSTM32V Dev. Board + 3.2" TFT LCD Module
- FTDI Chip UMFT311EV
- Fuente de alimentación Tektronix CPS250
- Osciloscopio PicoScope 3224

- 2 sondas CP-260 60 MHz
- Ordenador de sobremesa Sun Microsystems Intel Core 2
- Monitor Sun Microsystems 22" LCD

El periodo de amortización de un equipo electrónico está estipulado en 24 meses. Sin embargo la duración de este TFG ha sido de 4 meses, por lo que el coste del material será correspondiente a la sexta parte del precio de mercado de cada equipo. Los dispositivos comprados durante la ejecución del TFG no presentan periodo de amortización.

Tabla 4. Costes recursos hardware

Hardware	Precio	Periodo amortización	Periodo de uso	Importe
Muscle Sensor v3	34,15 €	-	4 meses	34.15 €
HY-MiniSTM32V	28,69 €	-	-	28,69 €
UMFT311EV	28,65 €	-	-	28,65 €
Tektronix CPS250	459 €	24 meses	4 meses	76,5 €
PicoScope 3224	590,66 €	24 meses	4 meses	98,44 €
Sondas CP-269 60 MHz	38,99 €	24 meses	4 meses	6,50 €
Ordenador sobremesa	854,85 €	24 meses	4 meses	142,48 €
Monitor	284,25 €	24 meses	4 meses	47,38 €
TOTAL		**462,79 €**		

6.1.2 Recursos Software

Los recursos software utilizados a lo largo del desarrollo del TFG son los siguientes:

- Keil µVision 4.73
- Android Studio
- Microsoft Office 2013

Al igual que con el material hardware, el material software también presenta un periodo de amortización de 24 meses, por lo que el importe será calculado como la sexta parte del precio de mercado.

Tabla 5. Costes recursos software

Hardware	Precio	Periodo amortización	Periodo de uso	Importe
Keil µVision 4.73	8.538,10 €	24 meses	4 meses	1423,02 €
Android Studio	0 €	-	-	0 €
Microsoft Office 2013	269 €	24 meses	4 meses	44,83 €
TOTAL		**1.467,85 €**		

6.2 Trabajo tarifado por tiempo empleado

En este Trabajo Fin de Grado se han invertido 300 horas en las tareas de preparación, desarrollo y documentación necesarias para la elaboración del mismo. El importe de las horas de trabajo utilizadas para la realización del proyecto se haya utilizando las recomendaciones del COIT, a diferencia de que el sueldo por hora normal de trabajo de 50 € y 65 € por hora especial. Entonces utilizando la fórmula que facilita el COIT, queda que:

$$H = C_t \cdot 50 \cdot H_n + C_t \cdot 65 \cdot H_n$$

Donde:

- H son los honorarios totales por el tiempo dedicado.
- H_n son las horas normales trabajadas (dentro de la jornada laboral).
- H_e son las horas especiales.
- C_t es un factor de corrección función del número de horas trabajadas.

Las horas trabajadas han sido 300 (5h · 60 días) y todas ellas en horario normal. Entonces, según el COIT el coeficiente C_t tiene un valor variable en función del número de horas empleadas de acuerdo con la siguiente tabla:

Tabla 6. Factor de corrección en función del número de horas invertidas

Horas empleadas	Factor de corrección
Hasta 36 horas	1,00
Desde 36 horas a 72 horas	0,90
Desde 72 horas a 108 horas	0,80
Desde 108 horas a 144 horas	0,70
Desde 144 horas a 180 horas	0,65
Desde 180 horas a 360 horas	0,60
Desde 360 horas a 540 horas	0.55

Atendiendo entonces a la Tabla 6 el factor de corrección C_t es de 0,60 debido a que las horas trabajadas se comprenden entre las 180 y 360 horas. De esta manera se queda la ecuación siguiente:

$$H = 0{,}6 \cdot 50 \cdot 300 + 0 = 9.000\ €$$

Los honorarios totales por tiempo dedicado libres de impuestos ascienden a *nueve mil euros* (**9.000 €**).

6.3 Costes de redacción del Trabajo Fin de Grado

El importe de la redacción del proyecto se calcula de acuerdo a la siguiente expresión:

$$R = 0{,}07 \cdot P \cdot C_n$$

Donde:

- P es el presupuesto.
- C_n es el coeficiente de ponderación según el presupuesto.

Para este cálculo es necesario saber el presupuesto hasta el momento del TFG, que se muestra en la Tabla 7.

Tabla 7. Presupuesto

Recursos	Costes
Recursos materiales	462,79 €
Recursos Software	1.467,85 €
Trabajo tarifado por tiempo empleado	9.000 €
TOTAL	**10.930,64 €**

El presupuesto calculado hasta el momento asciende a 10.930,64 €. Como el coeficiente de ponderación para presupuestos inferiores a 30.050 € viene definido por el COIT, con un valor de 1.00, el coste derivado de la redacción del Trabajo Fin de Grado es de:

$$R = 0{,}07 \cdot 10.930{,}64 \cdot 1{,}00 = 765{,}14\ €$$

6.4 Material fungible

Además de los recursos hardware y software, en este proyecto, se han utilizado otros materiales como son los folios, impresión y encuadernación.

Tabla 8. Costes material fungible

Materiales	Costes
Folios	12 €
Impresión	40 €
Encuadernación	9 €
TOTAL	**65 €**

6.5 Derechos de visado del COIT

Los gastos de visado del COIT, precios orientativos, se tarifan mediante la siguiente expresión:

$$V = 0{,}006 \cdot P \cdot C_V$$

Donde:

- P es el presupuesto del proyecto
- C_V es el coeficiente reductor en función del presupuesto del proyecto.

El presupuesto P, calculado hasta el momento corresponde a la suma de los costes de ejecución material, de redacción y de material fungible.

$$P = 1.467{,}85 + 65 + 10.930{,}64 = 12.463{,}49\ €$$

Como el coeficiente de ponderación para presupuestos menores de 30.050 € viene definido por el COIT con un valor de 1.00, el coste de los derechos de visado del proyecto asciende a la cantidad de:

$$V = 0{,}006 \cdot 12.463{,}49 \cdot 1.00 = 74{,}78\ €$$

Por tanto el coste de los derechos de visado del proyecto asciende a *setenta y cuatro euros con setenta y ocho céntimos* (**74,78 €**).

6.6 Gastos de tramitación y envío

Los gastos de tramitación y envío están fijados en 6.01 €.

6.7 Aplicación de impuestos

El coste total del proyecto, antes de aplicarle los correspondientes impuestos, asciende 18.523,72 €, a lo que hay que sumarle el 7% de IGIC, con lo que el coste definitivo del Trabajo Fin de Grado es:

Tabla 9. Presupuesto total del proyecto

RECURSOS MATERIALES	1.467,85 €
TRABAJO TARIFADO POR TIEMPO EMPLEADO	10.930,64 €
REDACCIÓN DEL TFG	765,14 €
MATERIAL FUNGIBLE	65 €
DERECHOS DE VISADO	74,78 €
GASTOS DE TRAMITACIÓN Y ENVÍO	6,01 €
SUBTOTAL	13.309,42 €
APLICACIÓN DE IMPUESTOS (IGIC 7%)	931,66 €
TOTAL PRESUPESTO	**14.241,08 €**

El presupuesto total asciende a la cantidad de *catorce mil doscientos cuarenta y un euros y ocho céntimos* (**14.241,08 €**).

Las Palmas de Gran Canaria a 14 de Diciembre del 2015.

Fdo.: Dailos Fernando Ramírez Guski

Bibliografía

[1] Oña, A., Martínez, M., Moreno, F., Serra, E., & Arellano, R. (1993). Optimización de los componentes temporales de la salida de atletismo a través del control de la información. *Revista de psicología del deporte*, *2*(1), 0005-015.

[2] Marcos Zuberoa, *Big Data y fútbol: Así aprovecha el Real Madrid la tecnología.* Artículo de El País. 7 de Diciembre de 2015.

[3] Edward Sazonov y Michael R. Neuman. "Wearable Sensors. Fundamentals, Implementation and Applications. 2014 Elsevier Inc. ISBN: 978-0-12-418662-0, p. 1-101.

[4] Jianqing Wang y Qiong Wang. "Body Area Communications. Channel modeling, Communication systems and Emic". 2013 John Wiley & Sons Singapore Pte. Ltd. ISBN 978-1-118-18848-4 (cloth).

[5] Jesús Merino Pérez y María José Noriega Borge, "Fisiología general", Universidad de Cantabria (UC), Open Course Ware.

[6] Latarjet, M., & Liard, A. R. (2005). *Anatomía humana* (Vol. 2). Ed. Médica Panamericana.

[7] J.W. Wilson K. y col. Tejido muscular. Cap 2. En Anatomía y fisiología en la salud y enfermedad. 4ª Ed. Ed. Manual Moderno, México 1994.

[8] ASEM Galicia – Asociación Gallega contra las enfermedades Neuromusculares, "El Músculo Esquelético", Saber & entender, Junio 2003.

[9] Merletti, R., & Parker, P. A. (2004). *Electromyography: physiology, engineering, and non-invasive applications* (Vol. 11). John Wiley & Sons.

[10] Gianikellis, K., Mariño, M. M., & Fernández, F. A. (2011). La Electromiografía (EMG) como método para determinar la intervención muscular en los Deportes de Precisión. *COLECCIÓN ICD: INVESTIGACIÓN EN CIENCIAS DEL DEPORTE*, (13).

[11] Gila, A. Malanda, I. Rodríguez Carreño, J. Rodríguez Falces, J. Navallas, "Métodos de procesamiento de análisis de señales electromiográficas", An. Sist. Sanit. Navar. 2009, Vol. 32, Suplemento 3.

[12] Disselhorst-Klug, C., Schmitz-Rode, T., & Rau, G. (2009). Surface electromyography and muscle force: limits in sEMG–force relationship and new approaches for applications. *Clinical biomechanics*, *24*(3),.

[13] Masuda, T., & Sadoyama, T. (1986). The propagation of single motor unit action potentials detected by a surface electrode array.*Electroencephalography and clinical Neurophysiology*, *63*(6), 590-598.

[14] Caballero, K., Duque, L. M., Ceballos, S., Ramirez, J. C., & Peláez, A. (2002). Conceptos básicos para el análisis electromiográfico. *CES Odontología*, *15*(1).

[15] L.E. Mendoza, R.D. Castellano G., R.D. Rojas. (2007). *Aportes y alcancees de las técnicas de procesamiento, clasificación y descomposición de señales electromiográficas.* Universidad de los Andes, Venezuela.

[16] Ruiz, A. F., Brunetti, F. J., Rocon, E., Forner-Cordero, A., Pons, J. L., & de Bioingeniería, G. (2007). Adquisición y procesado de información EMG en el modelado de sistemas biológicos. *Jornadas de Automática*.

[17] Harold A. Romo, Esp., JudyC. Realpe, ING., Pablo E. Jojoa, PhD (2007). *Análisis de Señales EMG Superficiales y su Aplicación en Control de Prótesis de Mano.* Universidad de Cauca

[18] Reaz, M. B. I., Hussain, M. S., & Mohd-Yasin, F. (2006). Techniques of EMG signal analysis: detection, processing, classification and applications.*Biological procedures online*, *8*(1), 11-35.

[19] Advancer Technologies, "Three-lead Differential Muscle/Electromyography Sensor for Microcontroller Applications", Muscle Sensor v3, 4 febrero 2013.

[20] Datasheet AD8221. Precision Instrumentation Amplifier. Analog Devices. Obtenido el 17 de Febrero de 2015: http://pdf1.alldatasheet.com/datasheet-pdf/view/100258/AD/AD8221.html

[21] Prof. A. Roldán Aranda. Amplificadores Operacionales, RECTIFICADOR. Departamento de Electrónica y Tecnología de Computadores. Universidad de Granada

[22] Amplificadores operacionales con diodos. Obtenido el 12 de Diciembre de 2015: http://www.geocities.ws/raguma_006/descargas/rectificador_de_presicion.pdf

[23] Hoja de características del microprocesador STM32F103VCT6. http://www.st.com/web/en/resource/technical/document/datasheet/CD00191185.pdf

[24] Yiu, J. (2009). *The definitive guide to the ARM Cortex-M3*. Newnes.

[25] Martin, T. (2008). The Insider's Guide to the STM32 ARM® Based Microcontroller. *An Engineer's Introduction to the STM32 Series, Version*, *1*, 1-103.

[26] STMicroelectronics. (2014). RM0008 Reference manual, STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced ARM®-based 32-bit MCUs.
Obtenido el 2 de Febrero de 2015 de:
http://www.st.com/web/en/resource/technical/document/reference_manual/CD00171190.pdf

[27] HAOYU Electronics. (2010). *Schematic HY-MiniSTM32V*.
Obtenido el 2 de Febrero de 2015 de:
http://www.haoyuelectronics.com//Attachment/HY-MiniSTM32V/HY-MiniSTM32V_SCH.pdf

[28] FTDI Chip. (2012) . DS_UMFT311EV USB Android Host Module Datasheet.
Obtenido el 9 de Febrero de 2015 de:
http://www.ftdichip.com/Support/Documents/DataSheets/Modules/DS_UMFT311EV.pdf

[29] Future Technology Devices International Limited (FTDI). (2012). Connecting Peripherals to an Android Platform.
Obtenido el 12 de Febrero de 2015 de:
http://www.ftdichip.com/Products/ICs/FT311D.html

[30] Hoja de características del FTDI Chip DS-UMFT311EV.
http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT311D.pdf

[31] Entorno de programación Keil µVision4
http://www.keil.com/support/man/docs/uv4/uv4_overview.htm

[32] Entorno de programación Android Studio
http://www.europapress.es/portaltic/software/noticia-google-lanza-android-studio-nuevo-entorno-programacion-android-20130516142704.html

[33] Gehnring, J. (2015). *Graph View*.
Obtenido el 19 de Marzo de 2015 de:
http://www.android-graphview.org/

Anexos

Anexo I.

En este anexo se presentará el esquemático completo de la placa HY-MiniSTM32V, donde se puede apreciar la asignación de cada uno de los pines y los componentes que lo forman.

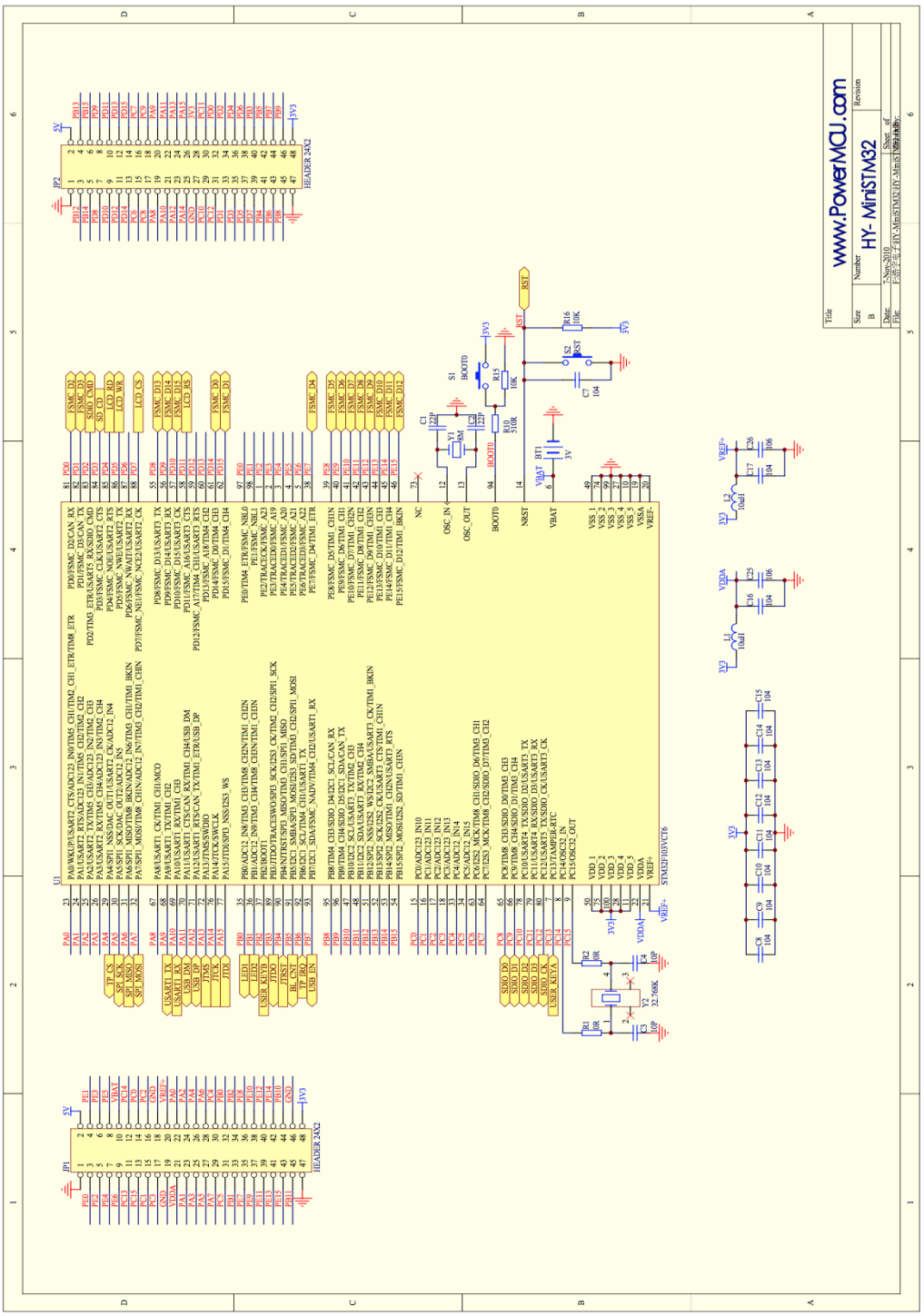

Figura 67. Esquemático configuración de pines del HY-MiniSTM32V

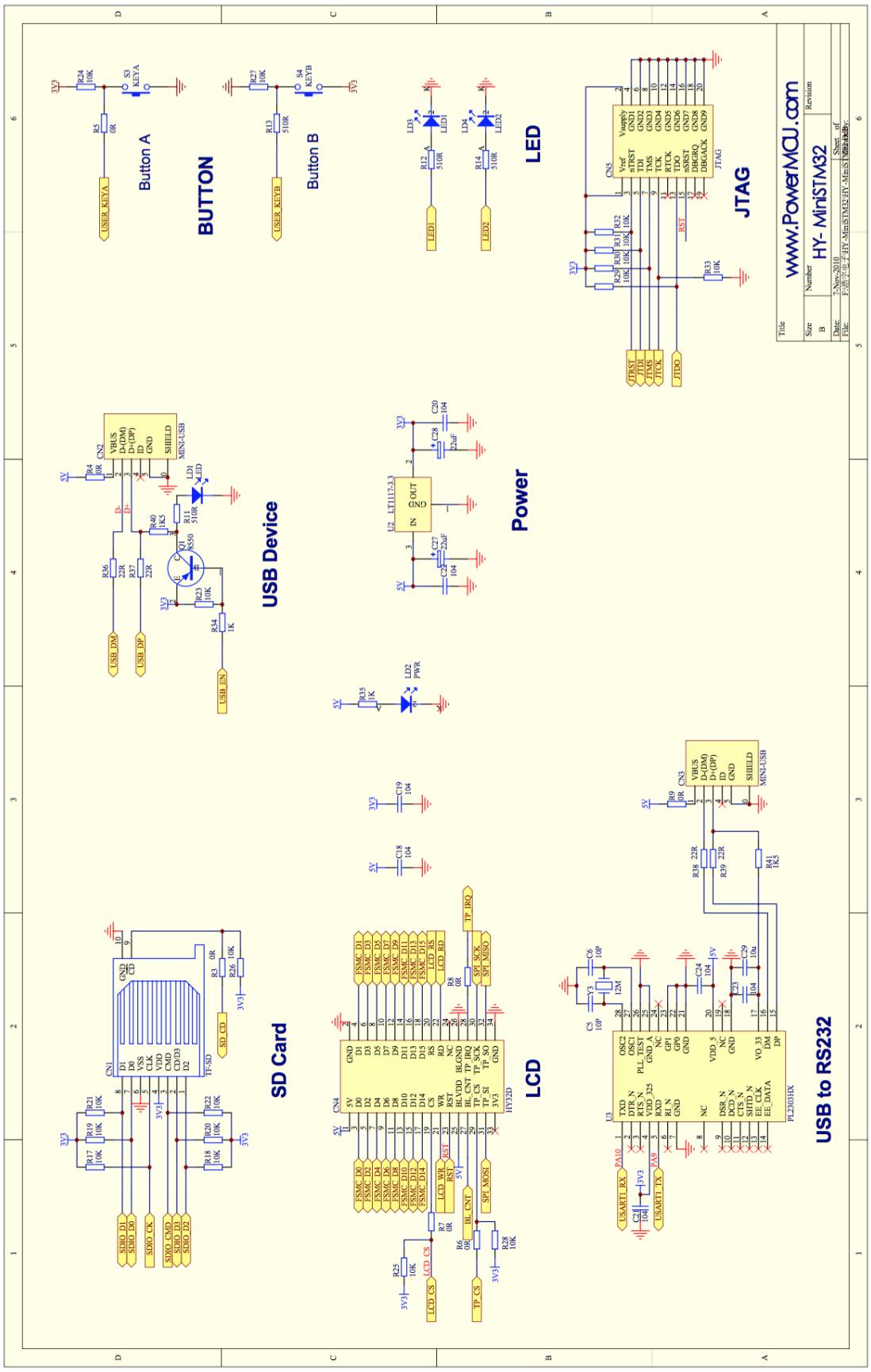

Figura 68. Esquemático de los componentes del HY-MiniSTM32V

Anexo II.

En el anexo 2 se halla el esquemático del módulo de desarrollo UMFT311EV. Indica la función de los distintos pines que forman parte del FT311D. Además señala las distintas configuraciones que se pueden realizar con los jumpers para obtener el modo de comunicación que se desee.

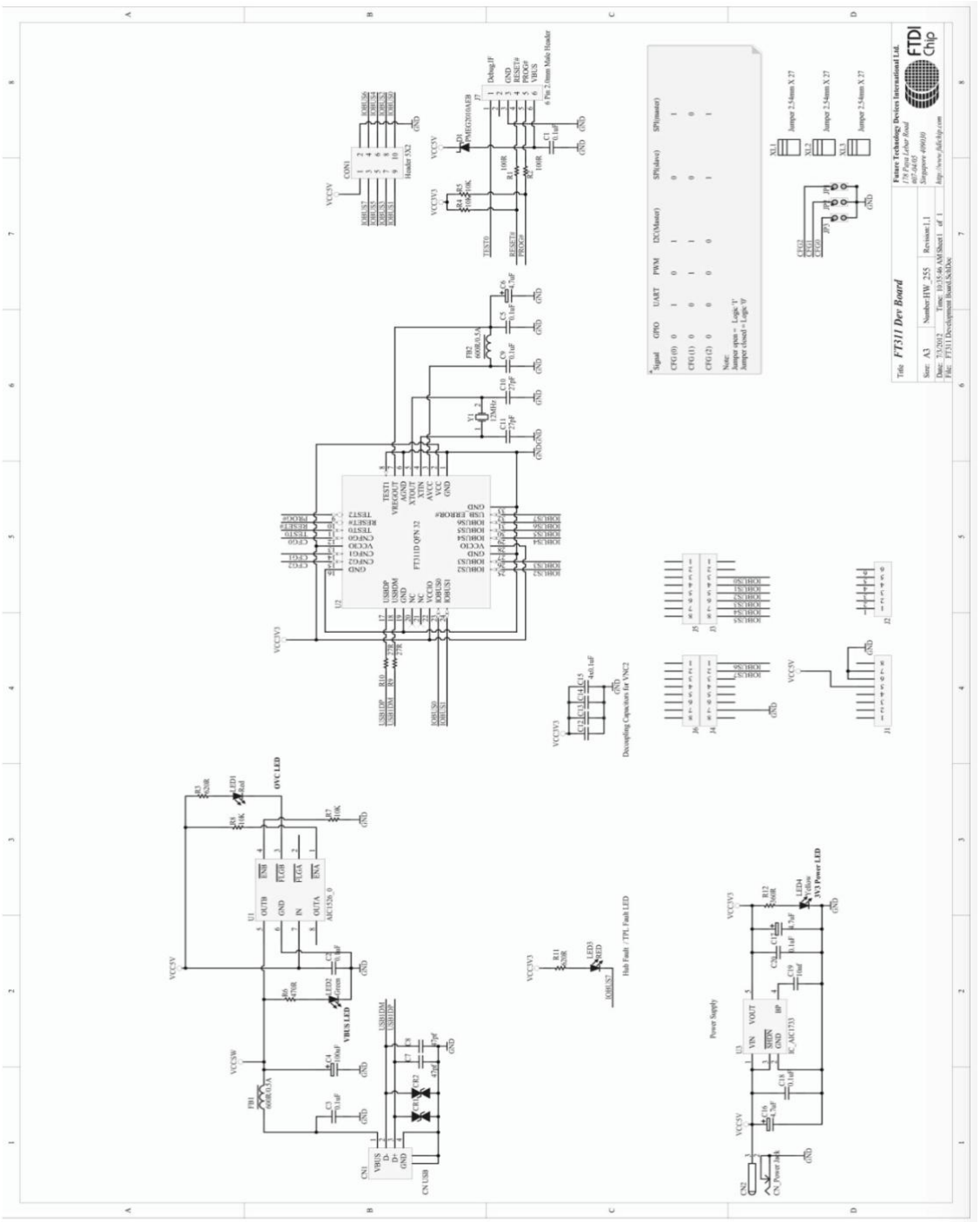

Figura 69. Esquemático FTDI Chip DS-UMFT311EV

Anexo III.

En este anexo se expondrá el código desarrollado durante el TFG. El código comprende tanto las tareas de adquisición de señal, como la de su representación y transmisión hacia el soporte Android. Este código pertenece al fichero *main.c*.

```
int main(void){
	static int i = 0;
	static int j = 0;
	static float cociente = 0;
	static float result;
	static uint16_t valor_pixel;
	static uint16_t dato;
	static uint16_t promedio = 0;
	static uint16_t aux = 0;
	uint8_t vuelta = 0;
	uint16_t x = 0;
	uint16_t datoAlto, datoBajo;
	uint8_t contador= 0;
	static uint8_t datoTx1, datoTx0;

	init_Buffer(&buffer2, 100);

	/* System clocks configuration ---------------------------------*/
	RCC_Configuration();

	/* NVIC configuration ------------------------------------------*/
	NVIC_Configuration();

	/* GPIO configuration ------------------------------------------*/
	GPIO_Configuration();

	/* TIM configuration -------------------------------------------*/
	TIM_Configuration();

	/* ADC configuration -------------------------------------------*/
	ADC_Configuration();

	/* TIM Enable --------------------------------------------------*/
	TIM_Enable();

	/* USART configuration -----------------------------------------*/
	USART_Configuration();

	/* GLCD configuration ------------------------------------------*/
	LCD_Initializtion();
	LCD_Clear(Blue);
```

```
/* TouchPanel Inicializacion ---------------------------------*/
TP_Init();
TouchPanel_Calibrate();
config_visual();
Retardo (10000000);
LCD_Clear(White);
cociente = (((float)233)/((float)4096));
calculo_Inicial(tiempo);
Ejes();

while (1)
{
      while(convertir == 0);  //Esta es la variable que cambia de
      valor una vez se entre en la ISR del timer1
      for (j = 0; j< 4; j++){
            ADC_SoftwareStartConvCmd(ADC1, ENABLE);
            /*Habilitamos el disparo por Software del ADC*/
            while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) ==
            RESET);
            aux = ADC_GetConversionValue(ADC1);/*Se coge el
            valor proporcionado por el ADC*/
            promedio = aux + promedio; /*Se va sumando a la
            variable promedio el valor que se va adquiriendo en
            cada momento del ADC*/
      }
      ADC_ClearITPendingBit(ADC1, ADC_IT_EOC);
      promedio = (promedio/j); /*Se calcula el promedio del total
      de unas 4 muestras que se adquieren casi simultáneamente */
      Ejes();
      push (&buffer2, promedio); /*Se mete en la cola circular el
      dato promediado en la parte superior de este código*/
      resultado = popFast(&buffer2);
      dato = *resultado;
      result = (((float)dato)*(cociente)); /*Se calcula el pixel
      a pintar*/
      valor_pixel = (uint16_t) result; /*en esta variable se
      guarda el pixel a pintar*/
      if(valor_pixel>=233){
            valor_pixel = 233 /*Forzamos a que el valor máximo de
            esta variable sea 233 para que en la resta no de
            menos de 2 en ningún momento, consiguiendo así el
            límite superior*/
      }
      valor_pixel = 235 - valor_pixel;
      almacen_Datos[i] = valor_pixel;
      actual = almacen_Datos[i];
      i++;
```

```
            if (i > 1){
                  if (vuelta > 0){
                        x = eje_x;
                        for (j = 0; j< 30; j++){
                        LCD_DrawLine(x, 2, x, 235, White);
                        x++;
                        }
                  }
                  anterior = almacen_Datos[i-2];
                  if(i == 100){
                        almacen_Datos[0] = almacen_Datos[i-1];
                        i = 1;
                  }
                  dibujar_Linea2(eje_x,actual, (eje_x-incremento_x),
                  anterior, Blue);
                  contador = 0;
            }
            eje_x = eje_x + incremento_x;
            if (eje_x >= 310){
                  eje_x = 10;
                  vuelta = 1;
            }

            /* Se realizan las operaciones necesarias para coger los
            primeros 6 bits de datos del ADC*/
            datoAlto = ((dato & 0xFC0) >> 6);
            datoAlto = (datoAlto | 0x40);
            datoTx1 = datoAlto;

            /* Se realizan las operaciones necesarias para coger los
            ultimos 6 bits de datos del ADC*/
            datoBajo = ((dato & 0x3F) | 0x80);
            datoTx0 = datoBajo;

            /* Enviamos por la USART1 los primeros 6 bits con cabecera
            0x40*/
            while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) ==
            RESET);
            USART_SendData(USART1, datoTx1);

            /* Enviamos por la USART1 los ultimos 6 bits con cabecera
            de 0x80*/
            while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) ==
            RESET);
            USART_SendData(USART1, datoTx0);

            /*Ponemos a 0 de nuevo la variable convertir para que el
            timer le vuelva a dar valor de 1 y se dispare el ADC*/
            convertir = 0;
      }
}
```

```
void Retardo (int cont){
      int j = cont;
      for( j = cont; j>0; j--);
}

/* Función para dibujar los ejes X e Y*/
void Ejes (void){
      uint16_t k;
      uint16_t p = 10;
      uint16_t x = 10;
      for(k = 0; k< 23; k ++){
            LCD_DrawLine(12,p,8,p,Black);
            p = p +10;
      }
      LCD_DrawLine (x,2, x, 235, Black);              //Eje y
      LCD_DrawLine(310,236,10,236, Black);            //Eje x
}
```

Anexo IV.

El código que se presentará a continuación corresponde al fichero en el que se realizan las configuraciones de aquellas prestaciones que se van a realizar. Este código pertenece al fichero *config.c*.

```
void RCC_Configuration(void)
{
	/* ADCCLK = PCLK2/6 */
	RCC_ADCCLKConfig(RCC_PCLK2_Div6); /*Poner Div6 implica que el
	ADCCLK vaya a 9.33 MHz. Se pone a esta frecuencia porque en el
	reference manual dice que tiene que ser como maximo 14 MHz*/
	/* Enable peripheral clocks ----------------------------------*/
	/* Enable DMA1 clock */
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE);
	/* Enable GPIOC, ADC1 and TIM1 clock */
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC |
	RCC_APB2Periph_ADC1, ENABLE);

	/*Enable TIM1 clock */
	RCC_APB1PeriphClockCmd (RCC_APB1Periph_TIM2, ENABLE);
	/* Forces or releases Low Speed APB (APB1) peripheral reset */
	//RCC_APB1PeriphResetCmd(RCC_APB1Periph_TIM2, ENABLE);
}

void GPIO_Configuration(void)
{
	GPIO_InitTypeDef GPIO_InitStructure;
	/* Configure PC.01 (ADC Channel11) as analog input */
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AIN;
	GPIO_Init(GPIOC, &GPIO_InitStructure);
}

void NVIC_Configuration(void)
{
	NVIC_InitTypeDef NVIC_InitStructure;
	/* Configure and enable TIM2 interrupt */
	NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
	NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
```

```
	NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
	NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
	NVIC_Init(&NVIC_InitStructure);
}

void TIM_Configuration(void)
{
	TIM_TimeBaseInitTypeDef   TIM_TimeBaseStructure;
	TIM_OCInitTypeDef         TIM_OCInitStructure;
	/* TIM1 configuration ----------------------------------------*/
	/* 72MHz/Prescaler(7200) = 10KHz -> 10KHz / Frecuencia deseada =
	Period */
	/* Time Base configuration */
	TIM_TimeBaseStructInit(&TIM_TimeBaseStructure);
	TIM_TimeBaseStructure.TIM_Period = 12 - 1; /*Muestreo a 833 Hz*/
	TIM_TimeBaseStructure.TIM_Prescaler = 7200 - 1;
	TIM_TimeBaseStructure.TIM_ClockDivision = TIM_CKD_DIV1;
	TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
	TIM_TimeBaseInit(TIM2, &TIM_TimeBaseStructure);
	/* TIM2 channel1 configuration in Compare mode */
	TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Active;
	TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
	TIM_OCInitStructure.TIM_Pulse = 1;
	TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
	TIM_OC2Init(TIM2, &TIM_OCInitStructure);
	TIM_ITConfig(TIM2, TIM_IT_CC2, ENABLE);
}

void TIM_Enable(void){
	/* TIM1 counter enable */
	TIM_Cmd(TIM2, ENABLE);
	/* TIM1 main Output Enable */
	TIM_CtrlPWMOutputs(TIM2, ENABLE);
}

void USART_Configuration(void)
{
	GPIO_InitTypeDef GPIO_InitStructure;
	USART_InitTypeDef USART_InitStructure;
	RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOA |
	RCC_APB2Periph_USART1,ENABLE);
```

```
	/*USART1_TX -> PA9 , USART1_RX ->  PA10 */
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOA, &GPIO_InitStructure);
	GPIO_InitStructure.GPIO_Pin = GPIO_Pin_10;
	GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;
	GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOA, &GPIO_InitStructure);
	USART_InitStructure.USART_BaudRate = 9600;
	USART_InitStructure.USART_WordLength = USART_WordLength_8b;
	USART_InitStructure.USART_StopBits = USART_StopBits_1;
	USART_InitStructure.USART_Parity = USART_Parity_No;
	USART_InitStructure.USART_HardwareFlowControl =
	USART_HardwareFlowControl_None;
	USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
	USART_Init(USART1, &USART_InitStructure);
	USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
	USART_ITConfig(USART1, USART_IT_TXE, ENABLE);
	USART_Cmd(USART1, ENABLE);
}

void ADC_Configuration(void){
	ADC_InitTypeDef ADC_InitStructure;
	/* ADC1 configuration ------------------------------------------*/
	ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
	ADC_InitStructure.ADC_ScanConvMode = DISABLE;
	ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
	ADC_InitStructure.ADC_ExternalTrigConv =
	ADC_ExternalTrigConv_None; //MODIFICADO
	ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
	ADC_InitStructure.ADC_NbrOfChannel = 1;
	ADC_Init(ADC1, &ADC_InitStructure);
	/* ADC1 regular channel11 configuration */
	ADC_RegularChannelConfig(ADC1, ADC_Channel_11, 1,
	ADC_SampleTime_1Cycles5);
	/* Enable ADC1 external trigger */
	ADC_ExternalTrigConvCmd(ADC1, DISABLE); //MODIFICADO

	/* Enable EOC interrupt */
	 ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);
```

```
    /* Enable ADC1 */
    ADC_Cmd(ADC1, ENABLE);
    /* Enable ADC1 reset calibration register */
    ADC_ResetCalibration(ADC1);
    /* Check the end of ADC1 reset calibration register */
    while(ADC_GetResetCalibrationStatus(ADC1));
    /* Start ADC1 calibration */
    ADC_StartCalibration(ADC1);
    /* Check the end of ADC1 calibration */
    while(ADC_GetCalibrationStatus(ADC1));
}
```

Anexo V.

El siguiente código que se presenta es el que corresponde al fichero *dibujar.c* y es el código que se ha desarrollado para la configuración del tiempo en el que queremos representar la señal.

```
void calculo_Inicial (uint16_t tiempo){
	uint16_t mps, pixeles; /* mps = muestras por segundo*/
	pixeles = 300;
	if(tiempo == 1 || tiempo == 2 || tiempo == 3 || tiempo == 6){
		if(tiempo == 6){
			incremento_x = 1;
		}else if( tiempo == 3){
			incremento_x = 2;
		}else if(tiempo == 2){
			incremento_x = 3;
		}else if(tiempo == 1){
			incremento_x = 6;
		}
	}else{
		GUI_Text(45,140, "Tiempo invalido", White, Blue);
	}
}

void config_visual(){
	uint16_t x, y, p;
	uint16_t oper = 0;
	uint8_t et1 = 0, et2 = 0, et3 = 0, et5 = 0, et6 = 0;
	LCD_Clear(Blue);
	GUI_Text(23,70,"Tiempo a mostrar:",White,Blue);
	LCD_CheckBox(20,90,0,Green);
	GUI_Text(45,90," 6 segundos ",Blue,White);
	LCD_CheckBox(20,110,0,Green);
	GUI_Text(45,110," 3 segundos ",Blue,White);
	LCD_CheckBox(20,130,0,Green);
	GUI_Text(45,130," 2 segundos ",Blue,White);
	LCD_CheckBox(20,150,0,Green);
	GUI_Text(45,150," 1 segundos ",Blue,White);
	while(oper == 0){
		p = LCD_TouchRead(&display); /lee la posición del puntero
		sobre la pantalla */
		x = display.x; /*extrae coordenada x*/
		y = display.y; /*extrae coordenada y*/
		if(p == 1) { /*detecta pulsación sobre la pantalla*/
			if( (et1==0) && (x>20) && (x<36) && (y>150) &&
			(y<176) ) {et1=1,oper=1;}
			if((et2==0) && (x>20) && (x<36) && (y>130) &&
			(y<156)){et2=1,oper=1;}
			if((et3==0) && (x>20) && (x<36) && (y>110) &&
			(y<126)){et3=1,oper=1;}
```

```
                if((et6==0) && (x>20) && (x<36) && (y>90) &&
                (y<106)){et6=1,oper=1;}
                if(oper==1)
                {
                /*indicación de "Activado" en etiquetas de los
                CheckBox*/
                      if(et1==1){
                            LCD_CheckBox(20,150,1,Green);
                            GUI_Text(45, 150, " 1 segundos " ,
                            Blue,White);
                            tiempo = 1;
                            oper = 1;
                            p = 0;
                      }
                      if(et2==1){
                            LCD_CheckBox(20,130,1,Green);
                            GUI_Text(45, 130, " 2 segundos ",
                            Blue,White);
                            tiempo = 2;
                            oper = 1;
                            p = 0;
                      }
                      if(et3==1){
                            LCD_CheckBox(20,110,1,Green);
                            GUI_Text(45, 110, " 3 segundos ",
                            Blue,White);
                            tiempo = 3;
                            oper = 1;
                            p = 0;
                      }
                      if(et6==1){
                            LCD_CheckBox(20,90,1,Green);
                            GUI_Text(45, 90, " 6 segundos ",
                            Blue,White);
                            tiempo = 6;
                            oper = 1;
                            p = 0;
                      }
                      LCD_Delay(500);
                }
          }
}
LCD_Clear(White);
}
```

Anexo VI.

En este anexo se presentará el código dedicado a la rutina de servicio de la interrupción que se produce cuando el timer ha llegado a su valor. Este código se encuentra en el fichero *stm32f10x_it.c*.

```
void TIM2_IRQHandler(void)
{
      if ( TIM_GetITStatus(TIM2 , TIM_IT_CC2) != RESET )
      {
            /* Start the conversion */
            convertir =1;
            /* Se limpia el flag de interrupcion */
            TIM_ClearITPendingBit(TIM2 , TIM_FLAG_CC2);
      }
}
```

Anexo VII.

El código desarrollado para la creación y manipulación de la cola circular es el que se encuentra en el fichero *buffer.c*.

```
void init_Buffer(circular *buffer, int size2){
	buffer->size = size2;
	buffer->start = buffer->buffer;
	buffer->end = &buffer->buffer[size2-1];
	buffer->readFast = buffer->buffer;
	buffer->write = buffer->buffer;
}

/* Comprobar si el buffer esta lleno */
int buffer_Full(circular *buffer){
	if((buffer->write >= buffer->end)
		return 1;
	}else{
		return 0;
	}
}

/* Comprobar si el buffer esta vacio */
int buffer_Empty(circular *buffer){
	if((buffer->write == buffer->start) && (buffer->readFast ==
	buffer->start)){
		return 1;
	}else{
		return 0;
	}
}

/*Insertamos dato en el buffer y en el caso de estar lleno se empieza
a insertar desde el principio */
void push(circular *buffer, uint16_t data) {
    if (buffer_Full(buffer)) {
		buffer->write = buffer->start;

    } else {
		*buffer->write = data;
		buffer->write++;
    }
}
```

```
/* Sacamos dato en el buffer y en el caso de leer la ultima posicion
se inicia desde el principio, y otro caso es que se iguale a la
escritura */
uint16_t * popFast(circular *buffer) {
    if (buffer_Empty(buffer)) {
            return buffer->end;
    } else {
            if (buffer->readFast >= buffer->end) {
                  buffer->readFast = buffer->start;
            }
                  return buffer->readFast++;
    }
}
```

Anexo VII.

La función que se encargar de representar la señal en el LCD es el que se expone a continuación. Se encuentra en el fichero *GLCD.c*.

```
void dibujar_Linea2(int x1, int y1, int x2, int y2,uint16_t bkColor)
{
	int x,y,dx,dy,Dx,Dy,e,i;
	Dx=x2-x1;
	Dy=y2-y1;
	dx=fabs(x2-x1);
	dy=fabs(y2-y1);
	x=x1;
	y=y1;
	if(dy>dx)
	{
		e=-dy;
		for(i=0;i<dy;i++)
		{
			LCD_SetPoint(x,y,bkColor);
			if(Dy>=0) y++;
			else y--;
			e+=2*dx;
			if(e>=0)
			{
				if(Dx>=0) x++;
				else x--;
				e-=2*dy;
			}
		}
	}
	else
	{
		e=-dx;
		for(i=0;i<dx;i++)
		{
			LCD_SetPoint(x,y,bkColor);
			if(Dx>=0) x++;
			else x--;
			e+=2*dy;
			if(e>=0)
			{
				if(Dy>=0) y++;
				else y--;
				e-=2*dx;
			}
		}
	}
}
```

Anexo VIII.

En este anexo se mostrará el código desarrollado para la aplicación Android.

```
public class MainActivityEMG extends Activity {

    private final Handler mHandler = new Handler();
    private LineGraphSeries<DataPoint> mSeries;
    private double graph2LastXValue = 0d;

    /* thread to read the data*/
    public handler_thread handlerThread;

    /* declare a FT311 UART interface variable */
    public FT311UARTInterface uartInterface;

    /* local variables */
    byte[] writeBuffer;
    byte[] readBuffer;
    int[] actualNumBytes;
    byte status;

    /*UART configuration variables*/
    int baudRate;
    byte stopBit;
    byte dataBit;
    byte parity;
    byte flowControl;
    public Context global_context;
    public boolean bConfiged = false;

    /*Play and Stop*/
    boolean running = true;

    /*Data received from UART*/
    int[] dataReceived = new int[4096];
    int read = 0;
    int write = 0;

    String configuration;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Intent intent = getIntent();
        configuration = intent.getStringExtra
        (Menu_activity.EXTRA_MESSAGE);

        setContentView(R.layout.main_activity_emg);
        global_context = this;
```

```
        /* allocate buffer */
        writeBuffer = new byte[64];
        readBuffer = new byte[4096];
        actualNumBytes = new int[1];

        init_UART();
        init_Graph();

        handlerThread = new handler_thread(handler);
        handlerThread.start();
    }

    @Override
    protected void onResume() {
        // Ideally should implement onResume() and onPause()
        // to take appropriate action when the activity looses focus
        super.onResume();
        if (2 == uartInterface.ResumeAccessory()) {
            baudRate = 9600;
            stopBit = 1;
            dataBit = 8;
            parity = 0;
            flowControl = 0;
        }
        Runnable mTimer = new Runnable() {
            @Override
            public void run() {
                if (running) {
                    for(int i=0; i<30;i++){
                        graph2LastXValue += 1d;
                        mSeries.appendData(new
DataPoint(graph2LastXValue, generateData()), true, 900);
                    }
                }
                mHandler.postDelayed(this, 100);
            }
        };
        mHandler.postDelayed(mTimer, 100); //tiempo que tarda en
empezar a dibujar
    }

    @Override
    protected void onPause() {
        super.onPause();
    }

    @Override
    protected void onStop() {
        super.onStop();
    }

    @Override
```

```
    protected void onDestroy() {
        uartInterface.DestroyAccessory(bConfiged);
        super.onDestroy();
    }

    final Handler handler = new Handler() {
        @Override
        public void handleMessage(Message msg) {

            for (int i = 0; i < actualNumBytes[0]; i++) {
                dataReceived[write] = Integer.valueOf((readBuffer[i] &
0xff));
                write++;
                if (write == 4096) {
                    write = 0;
                }
            }
        }
    };

    /* usb input data handler */
    private class handler_thread extends Thread {
        Handler mHandler;
        /* constructor */
        handler_thread(Handler h) {
            mHandler = h;
        }
        public void run() {
            Message msg;
            while (true) {
                try {
                    Thread.sleep(200);
                } catch (InterruptedException e) {
                }
                status = uartInterface.ReadData(4096, readBuffer,
actualNumBytes);

                if (status == 0x00 && actualNumBytes[0] > 0) {
                    msg = mHandler.obtainMessage();
                    mHandler.sendMessage(msg);
                }
            }
        }
    }

    private double generateData() {
        double result = 0;
        int dataFirstFrame = dataReceived[read];
        int dataSecondFrame = dataReceived[read + 1];
        int firstFrameMask = 0x40;
        int secondFrameMask = 0x80;

        if ((dataFirstFrame & firstFrameMask) != firstFrameMask) {
```

```
            read++;
        } else if ((dataFirstFrame & firstFrameMask) ==
firstFrameMask) {
            dataFirstFrame = ((dataFirstFrame & 63) << 6);
            if ((dataSecondFrame & secondFrameMask) ==
secondFrameMask) {
                dataSecondFrame = (dataSecondFrame & 63);
            }
            read += 2;

            if (read == 4096) {
                read = 0;
            }
            result = (double) (dataFirstFrame | dataSecondFrame);
        }

        return result;
    }

    public void init_UART() {
        baudRate = 9600; /* baud rate */
        stopBit = 1; /* 1:1stop bits, 2:2 stop bits */
        dataBit = 8;/* 8:8bit, 7: 7bit */
        parity = 0; /* 0: none, 1: odd, 2: even, 3: mark, 4: space */
        flowControl = 0; /* 0:none, 1: flow control(CTS,RTS) */
        uartInterface = new FT311UARTInterface(this);
        uartInterface.SetConfig(baudRate, stopBit, dataBit, parity,
flowControl);
    }

    public void init_Graph() {
        GraphView graph = (GraphView) findViewById(R.id.graph);
        mSeries = new LineGraphSeries<DataPoint>();
        graph.addSeries(mSeries);
        mSeries.setColor(Color.GREEN);

        graph.getViewport().setScrollable(true);
        graph.getViewport().setScalable(true);
        graph.getViewport().setYAxisBoundsManual(true);
        graph.getViewport().setXAxisBoundsManual(true);
        graph.getViewport().setMinY(0);
        graph.getViewport().setMaxY(4095);
        graph.getViewport().setMinX(0);
        graph.getViewport().setMaxX(900);

        graph.setBackgroundColor(0xFF0F3B05);

        graph.getGridLabelRenderer().setGridColor(0x4490EE90);

graph.getGridLabelRenderer().setHorizontalLabelsColor(0xFF0F3B05);

graph.getGridLabelRenderer().setVerticalLabelsColor(0xFF0F3B05);
        graph.getGridLabelRenderer().setNumHorizontalLabels(27);
```

```
        graph.getGridLabelRenderer().setNumVerticalLabels(15);
        graph.getGridLabelRenderer().setVerticalLabelsVisible(false);
        graph.getGridLabelRenderer().setHighlightZeroLines(false);

    }

    public void playAndStop(View view) {
        GraphView graph = (GraphView) findViewById(R.id.graph);
        ToggleButton button_playStop = (ToggleButton)
findViewById(R.id.button_playStop);
        boolean on = button_playStop.isChecked();
        if (!on) {
            running = false;
            graph.getViewport().setScalable(true);
        } else {
            running = true;
        }
    }
}
```