
Universidad de Valladolid

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA DE TECNOLOGÍAS ESPECÍFICAS DE TELECOMUNICACIÓN
MENCIÓN EN SISTEMAS ELECTRÓNICOS

# Desarrollo de un osciloscopio digital en Android

Autor:
**David Ortiz de Latierro Delgado**

Tutores:
**Iván Santos Tejido**
**Miguel Ángel González Rebollo**

Valladolid, Septiembre de 2019

| | |
|---|---|
| TÍTULO: | **Desarrollo de un osciloscopio digital en Android** |
| AUTOR: | **D. David Ortiz de Latierro Delgado** |
| TUTORES: | **D. Iván Santos Tejido** |
| | **D. Miguel Ángel González Rebollo** |
| DEPARTAMENTOS: | **Departamento de Electricidad y Electrónica**<br>**Departamento de Física de la Materia Condensada, Cristalografía y Mineralogía** |

**TRIBUNAL**

| | |
|---|---|
| PRESIDENTE: | **D. Jesús M. Hernández Mangas** |
| VOCAL: | **D. Iván Santos Tejido** |
| SECRETARIO | **D. Jesús Arias Álvarez** |
| SUPLENTE | **Dña. Lourdes Pelaz Montes** |
| SUPLENTE | **D. Pedro López Martín** |

| | |
|---|---|
| FECHA: | **Septiembre de 2019** |
| CALIFICACIÓN: | |

# Resumen

Este Trabajo de Fin de Grado se centra en el diseño, fabricación e implementación de un osciloscopio de bajo coste que utiliza un dispositivo con sistema operativo Android para mostrar, analizar y controlar las señales de interés capturadas.

El objetivo fundamental es que los centros de enseñanza y formación profesional puedan disponer de equipos de instrumentación electrónica más económicos que los comerciales, de manera que se pueda disponer de un pequeño laboratorio móvil y de bajo coste para la realización de medidas básicas de señales eléctricas alternas.

Tras analizar las diferentes alternativas posibles durante las prácticas en empresa, se optó por utilizar un DSP de la empresa *Microchip Technology* para capturar y adaptar las señales, y el uso del protocolo USB como método de interconexión entre nuestro dispositivo Android y el sistema con el DSP mencionado.

# Palabras clave

Osciloscopio, Android, DSP, USB, Microchip Technology, dsPIC33EP256GP504, C, Java, XML, MPLAB X IDE, Proteus, Android Studio.

# *Abstract*

In this TFG we focus on the design, manufacture and implementation of a low-cost oscilloscope which use an Android device in order to show, analyze and manage signals that we read.

The main goal is the fact that educative centers could have more economical electronic devices than commercial ones, so they could have a mobile and low-cost laboratory to measure basic alternate electric signals.

After analyzing the different alternatives during the professional practices, we decided the use of a DSP by Microchip Technology Inc. to capture and adapt the signals, and the use of USB protocol as an interconnection method between our Android device and the DSP.

# *Keywords*

Oscilloscope, Android, DSP, USB, Microchip Technology, dsPIC33EP256GP504, C, Java, XML, MPLAB X IDE, Proteus, Android Studio.

# Agradecimientos

Me gustaría expresar mi agradecimiento a todas las personas que han hecho posible la culminación de este trabajo.

A mis profesores del Grado, que durante los años de estudio me ayudaron a dibujar mi recorrido profesional. En particular a Iván Santos Tejido, por su apoyo e ideas para tratar de resolver las dificultades que se presentaban durante el proyecto, y a Jesús M. Hernández Mangas, por su apoyo y orientación en los momentos de mayor bloqueo, y por avivar mi interés en el campo de la electrónica.

Mi agradecimiento más grande para mis padres y hermana, y extensivamente a toda mi familia, por su gran paciencia y comprensión, por su apoyo y cariño incondicional, y por servirme de ejemplo para saber que puedo con todos los retos que surjan en vi vida personal y profesional.

Gracias a todos.

# Índice General

# Índice de Figuras

# Índice de Tablas



# Índice de Ecuaciones

# Capítulo 1. Introducción

## 1.1. Motivación

Dado que el trabajo experimental tanto en ciencia como en tecnología es una herramienta clave para el aprendizaje, se hace imprescindible el uso de laboratorios de prácticas en estas disciplinas. Sin embargo, no siempre es posible disponer de las instalaciones o los medios adecuados para que los estudiantes puedan aprender y desarrollar nuevas capacidades. Es por este motivo por el que surgen grupos de investigación como el GIR de Nuevas Tecnologías para la mejora de la calidad de la enseñanza.

El objetivo fundamental del grupo está centrado en estudiar como las *TICs* (Tecnologías de la Información y la Comunicación) pueden facilitar y hacer accesible al mayor número posible de estudiantes esta estrategia basada en el aprendizaje práctico. Para lograrlo se hace necesario el desarrollo de entornos y herramientas de bajo coste que permitan el diseño de prácticas docentes, tanto en entornos formales (laboratorios de prácticas), como en entornos informales.

Una de estas herramientas es el osciloscopio, un instrumento a menudo utilizado para visualizar, medir y analizar tensiones variables en el tiempo. Aunque, debido a la versatilidad de este tipo de instrumentos de medida, se utilizan para muchos otros fines y en muchas otras ramas de trabajo, ya que, provistos de un transductor adecuado que convierta la magnitud física de interés en señales eléctricas, podemos medir desde el ritmo cardíaco de un paciente, hasta el nivel de vibraciones en un coche, el valor de una presión o la potencia de un determinado sonido.

Los osciloscopios son instrumentos de visualización gráfica que muestran señales eléctricas variables en el tiempo. Aunque son utilizados a menudo acompañados de un analizador de espectros en electrónica de la señal, son dispositivos básicos para el trabajo experimental tanto en ciencia como en tecnología.

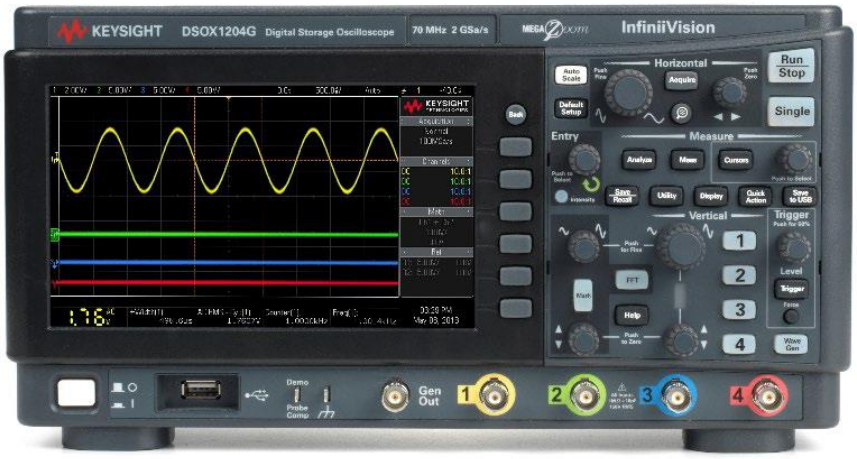



*Figura 1.1. Osciloscopio de 4 canales DSOX1204G (Keysight Technologies)*

El osciloscopio presenta los valores de las señales eléctricas de interés medidas en una pantalla, la cual muestra en su eje vertical (a menudo denominado Y) el voltaje, y en su eje horizontal (denominado X) el tiempo. El uso que se le suele dar a estos dispositivos es el siguiente:

- Determinar el periodo (o la frecuencia) y el voltaje de una señal.
- Determinar las componentes DC (Corriente Continua) y AC (Corriente Alterna) de una señal.
- Localizar averías en un circuito.
- Medir desfases entre señales.
- Determinar la componente de ruido de una señal y analizar su variación con el tiempo.

Inicialmente los primeros osciloscopios eran analógicos y estaban basados en el tubo de rayos catódicos (*CRT*), lo que les otorgaba el nombre de *CRO* (*Cathode Ray Oscilloscope*). Para su funcionamiento se hacía que el chorro de electrones del tubo se dirigiese hacia una pantalla fluorescente, controlando esta desviación mediante unas placas deflectoras que formaban parte de los sistemas de desviación vertical y horizontal a los cual se aplicaba la señal a medir y la señal de barrido respectivamente. Este tipo de osciloscopios podían representar la amplitud de la señal de entrada frente al tiempo, o frente a la amplitud de otra señal (modo X-Y). Se utilizan principalmente cuando queremos ver "en tiempo real" señales que varían rápidamente con el tiempo.

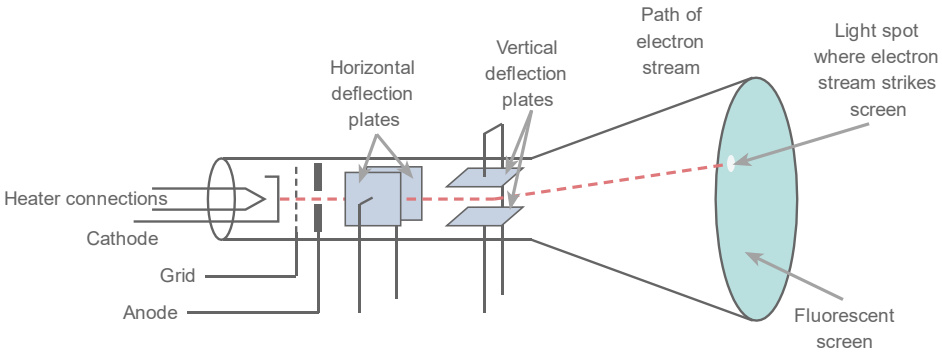



*Figura 1.2. Diagrama de un osciloscopio analógico que usa CRT*

Años más tarde, sobre 1985, llegaron los osciloscopios digitales (*DSO*, *Digital Storage Oscillocopes*). Estos osciloscopios muestrean la señal de entrada y almacenan la información hasta tener un número de puntos suficientes para poder representarla, lo que hace que la representación no sea en tiempo real. En este caso, la velocidad de actualización de la pantalla está limitada por el procesamiento de las muestras. Son adecuados para visualizar señales de baja frecuencia, transitorios y la parte de la señal anterior al punto de disparo.

La mayoría de osciloscopios de este tipo están basados en una *FPGA* (*Field Programmable Gate Array*) como elemento principal, debido a su alta velocidad para

controlar el muestreo y conversión analógica-digital, además del almacenamiento en memoria y el control de los buffers entre otras cosas.

Aunque, como se ha comentado, existen dos tipos de osciloscopios: analógicos y digitales, nosotros solo nos centraremos en los segundos, ya que además de ser el objeto de este trabajo, han ido desplazando a los primeros entre otras razones por la facilidad de poder transferir las medidas a un ordenador o a una pantalla LCD y son los más extendidos en la actualidad.

## 1.2. Objetivos

Durante la realización de este proyecto, tratamos de desarrollar un sistema que logre la misma funcionalidad que los osciloscopios tradicionales a un precio mucho más reducido, de manera que permita a un gran número de estudiantes e instituciones realizar prácticas docentes de una manera más económica y sencilla. Para conseguirlo se llevarán a cabo las tareas de diseño, fabricación, montaje y programación de una placa para la adquisición de datos y la programación de una App para su uso en un *Smartphone* o *Tablet* con sistema operativo *Android* para la representación de esos datos.

### 1.2.1. Hardware

Para la adquisición de las señales, se desarrollará una placa de circuito impreso (*PCB*) personalizada, que contendrá principalmente dos partes: una primera parte que consistirá en los circuitos necesarios para la captura y acondicionamiento de las señales a medir, para poder adaptarlas al conversor analógico-digital, y una segunda parte que se encargará de muestrear las señales y prepararlas para su envío hacia el dispositivo *Android*.

### 1.2.2. Software

Para la representación de las señales medidas se utilizará un dispositivo con sistema operativo *Android*, para el cual se programará una App en lenguaje *Java* que reciba los datos capturados, los interprete y los represente en la pantalla de la forma más fiable posible. Además, integrará una serie de controles para configurar la toma de muestras y la representación de las señales, e incluso para poder realizar algunas funciones con los datos recibidos.

## 1.3. Especificaciones previas

Una vez descrito el objetivo principal del proyecto, podemos particularizar un poco más en cuáles son las especificaciones propuestas inicialmente para nuestro sistema, donde encontramos que:

- Como la aplicación será desarrollada para un ámbito docente a nivel de Bachillerato y primeros cursos Universitarios, se hace necesario disponer de al menos 2 canales de entrada, para poder realizar operaciones sencillas o comparar diferentes señales.

- Aunque el rango de frecuencias de las señales a analizar no deberá ser muy grande, probablemente superará las decenas de KHz, por lo que se hace necesario tener un rango de análisis adecuado a estas frecuencias.

- En el caso de necesitar una depuración a la hora de resolver problemas de mal funcionamiento del sistema, un circuito con menor número de componentes será más fácil de analizar y resolver. Además, con circuitos más simples, el diseño y fabricación del mismo será más sencillo, por lo que se deberá elegir cuidadosamente los componentes a utilizar.

- Por otra parte, y en línea con lo anterior, cuanto menos complicado sea el circuito, más económico será el sistema final y por tanto mayor número de ellos se podrá conseguir.

Todos estos aspectos fueron analizados durante las prácticas en empresa (ver Anexo A), donde se buscaron diferentes alternativas para el desarrollo del sistema, así como diferentes métodos de interconexión entre la *PCB* y el dispositivo *Android*. Finalmente se decidió que la solución que se desarrollaría en este trabajo sería la realización de un osciloscopio digital conectado por USB al dispositivo *Android*, donde la captura y procesamiento de las señales a analizar se realizaría mediante un *DSP* (Procesador de Señales Digitales) externo.

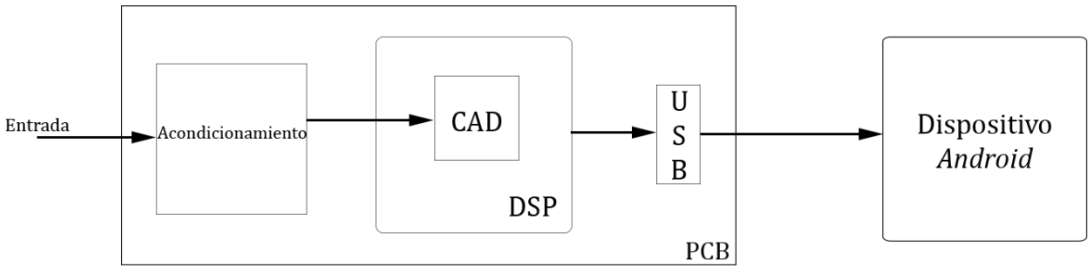



*Figura 1.3. Esquema de bloques del diseño final*

## 1.4. Estructura de la memoria

Tras obtener una descripción representativa de cuál será el objeto de este trabajo pasamos a presentar la organización de los posteriores capítulos del mismo.

En primer lugar, se describirán las partes principales de este tipo de sistemas, comparándolas con las partes de nuestro sistema. Además, se analizarán ciertas consideraciones que se deben tener en cuenta a la hora de diseñar el sistema o, una vez diseñado, en el momento de su utilización y comprensión.

En los capítulos siguientes se describirán paso por paso las diferentes partes que componen nuestro sistema. Comenzaremos con el Hardware, detallando cada una de las etapas por las que está formado, así como la fabricación y el montaje del mismo. A continuación, se explicará el Firmware (la programación del Hardware), que es el programa que se encargará de tomar las muestras, procesarlas y enviarlas al receptor. Y por último el Software (la programación de la App para *Android*), el cual recibirá los datos de las muestras, las reconstruirá y mostrará las señales captadas inicialmente. También notificará al Hardware sobre el modo de funcionamiento del mismo.

Durante los últimos capítulos del trabajo se comentarán las pruebas realizadas, así como los resultados obtenidos, se detallarán los costes totales del proyecto, y finalmente, se expondrá cuáles han sido las conclusiones, además de definir el posible trabajo futuro y las mejoras a realizar en una versión posterior del sistema.

# Capítulo 2. Descripción del sistema

En este capítulo, se describirán los requerimientos previos y las consideraciones que se deben tener en cuenta a la hora de diseñar el osciloscopio para el sistema de estudio.

## 2.1. Consideraciones específicas

Existen numerosos aspectos importantes que debemos tener en cuenta a la hora de decidir el/los modos de funcionamiento del dispositivo, así como comprender sus limitaciones más importantes y el porqué de las mismas.

### 2.1.1. Muestreo

El muestro de la señal de entrada es uno de los puntos más importantes y que más influyen en decidir cuál será el modo de funcionamiento de nuestro dispositivo. De manera que deberemos concluir que método usar según sea la frecuencia de la señal de entrada comparada con lo rápido que es nuestro sistema en tomar las muestras. Para decidir qué método utilizar según cada señal, podemos hacer uso del criterio de *Nyquist*. Este teorema establece que para poder reconstruir fielmente una señal de frecuencia $F_S$, debemos muestrear al menos al doble de la frecuencia de la señal, es decir, a $2F_S$. Si esto no se cumple, aparecen frecuencias falsas (*aliasing*), que son la diferencia entre la frecuencia de la señal y frecuencia de muestreo.

Por otra parte, el aspecto de la señal reconstruida dependerá también del método de interpolación utilizado, siendo los más comunes el método lineal y el basado en la función $\frac{\sin x}{x}$.

Existen principalmente dos métodos de muestreo:

- **Muestreo directo/tiempo real (*real time sampling*)**

Mediante este tipo se muestrean puntos consecutivos de la señal tal como viene, de manera que cada muestra y el tiempo en el que fue tomada tienen una correspondencia directa con su equivalente en tiempo real. Por este motivo, es el único tipo de muestreo que nos permite digitalizar completamente señales no periódicas, señales muy rápidas o transitorios.

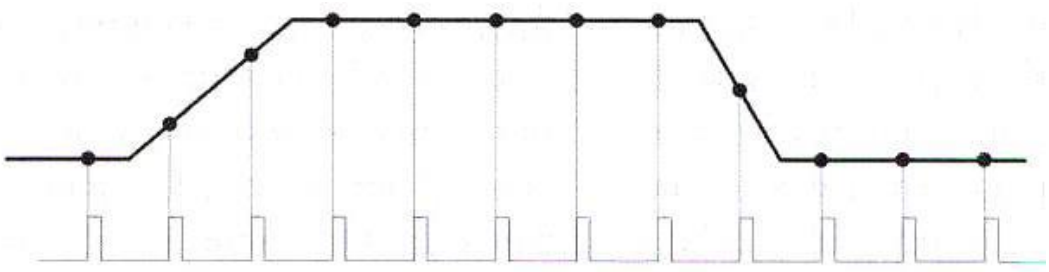


*Figura 2.1. Ejemplo de muestreo directo*

Se basan en el uso de CADs muy rápidos y en la interpolación entre muestras, y son útiles cuando la frecuencia de la señal de entrada es menor que el doble de la máxima velocidad de muestreo.

En la práctica se ha demostrado que muestreando a una frecuencia cuatro veces mayor que la mayor componente en frecuencia de la señal medida, se obtiene un resultado mucho más fiable y cercano al real con un error muy bajo. De modo que a medida que nos acercamos a la frecuencia de $2F_S$ encontramos más errores en la medida.

- **Muestreo en tiempo equivalente/repetitivo (*equivalent time sampling*)**

Con este otro método muestreamos de manera secuencial, tomando una muestra de cada uno de los ciclos sucesivos de la señal, lo que nos permite muestrear señales de frecuencias más elevadas. Aunque como contraparte, estas señales deben ser señales repetitivas, de manera que puedan ser reconstruidas a partir de las muestras tomadas en los diferentes ciclos.

Dentro de este tipo de muestreo podemos encontramos distintas variantes:

- **Muestreo aleatorio**

Durante el muestreo aleatorio tomamos una muestra en cada ciclo en un instante escogido de manera aleatoria con respecto a un punto de referencia de la señal de entrada. Tras haber muestreado varios ciclos, los puntos obtenidos se juntan componiendo la forma de onda final.

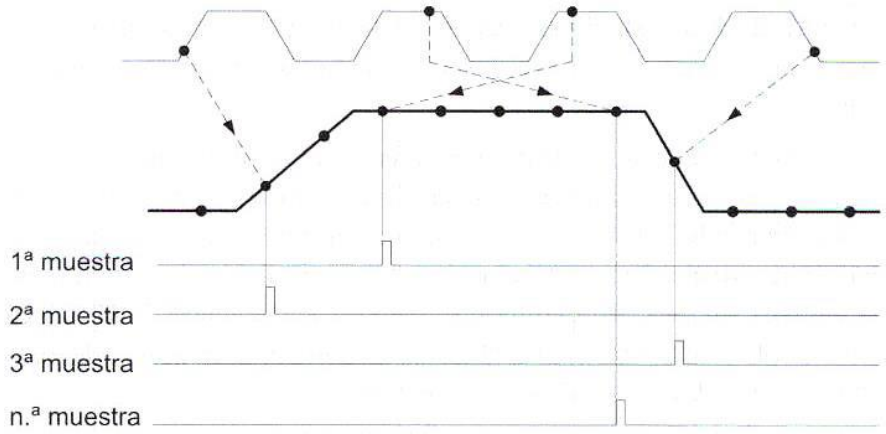



*Figura 2.2. Ejemplo de muestreo aleatorio*

A medida que se muestrean más ciclos obtenemos un mayor número de muestras, que luego, al ser colocadas en un mismo ciclo, nos permiten reconstruir de manera más precisa la señal.

- **Muestreo secuencial**

Este otro tipo es similar al anterior, salvo que en este caso se toma una muestra en cada ciclo en un instante que se incrementa a intervalos regulares con respecto a un punto de referencia de la señal de entrada. Para este tipo de muestreo es necesario, al igual que en el caso anterior, una señal periódica.

Las muestras son tomadas siempre en puntos diferentes del ciclo de la señal de entrada, sin importar en qué ciclo fueron tomadas. De manera que, al finalizar la captura de todas las muestras, cada una de ellas es posicionada en un único ciclo, pero en la posición que le correspondería según el tiempo que se incrementó desde el inicio de la captura.

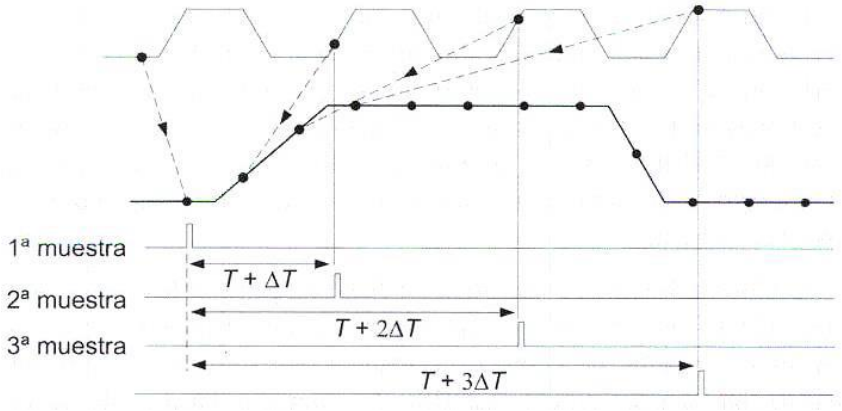



*Figura 2.3. Ejemplo de muestreo secuencial*

Una de las mayores ventajas de este tipo de muestreo es la posibilidad de reducir la velocidad del CAD, favoreciendo una mejor precisión.

En nuestro caso, el microprocesador escogido para el diseño tiene dos conversores independientes con la capacidad de muestrear a 1.1 Msps (aproximadamente un millón de muestras por segundo), con lo que seremos capaces de muestrear en tiempo real señales de hasta 500 KHz, lo cual está en el rango aceptable para el proyecto.

### 2.1.2. Sensibilidad y digitalización

La digitalización de la señal consiste en muestrear la señal en un determinado instante (*sample*), retener su valor (*hold*) y convertirlo mediante un CAD, pero existe un compromiso entre la velocidad de muestreo que determina la resolución temporal y la capacidad de la memoria de adquisición. Por ello, debemos elegir el tipo de conversión y posterior transferencia que mejor se adapte en cada ocasión.

Si convertimos la señal de forma continua, la señal se va muestreando y procesando de manera continua y sin interrupciones limitando su velocidad por la tasa de muestreo del CAD y la velocidad de transferencia de los datos al dispositivo que los represente. No habrá problemas para señales lentas o medir transitorios.

Sin embargo, si la vamos convirtiendo por ráfagas, lo que tendremos que hacer será tomar muestras de la señal hasta llenar un buffer, detener el proceso de muestreo mientras procesamos las muestras y volver a iterar una y otra vez. En este caso la velocidad vendrá limitada por la velocidad del propio microcontrolador y por la de transmisión hacia el dispositivo.

En la actualidad, los CADs existentes nos permiten obtener un número de muestras muy superior al de puntos que caben en la pantalla del dispositivo que representa

las señales medidas. Los modos de adquisición determinan la forma en que se obtienen los puntos a visualizar en la pantalla a partir de las muestras tomadas.

En cuanto a la sensibilidad (resolución en voltaje), esta viene determinada por el número de bits del CAD que estemos utilizando, y se determina según la siguiente fórmula:

$$Resolución = \frac{Tensión\ máxima\ de\ entrada\ al\ CAD}{2^n - 1}$$

*Ecuación 2.1. Cálculo de la resolución del CAD*

Donde 'n' es el número de bits del CAD.

En el caso de estudio, el microcontrolador encargado de realizar el muestreo, puede alcanzar resoluciones de hasta 12 bits, pero a una frecuencia de muestreo menor (500 Ksps). Para la frecuencia de muestreo más rápida (1,1 Msps), obtendremos una resolución de 10 bits.

### 2.1.3. Disparo (*Trigger*)

En un osciloscopio digital la captura de la señal no está sincronizada con una base de tiempos, sino que el disparo es una referencia temporal de las muestras tomadas. El mecanismo de disparo es el que nos permite que cualquier señal de entrada se represente en pantalla de forma estable o sincronizada. Es decir, es el sistema encargado de que la señal que vemos en la pantalla se inicie en su extremo izquierdo en un determinado momento para un determinado valor de la señal.

En los osciloscopios existen varias maneras de elegir esta referencia, denominados modos de disparo. Podemos ver los más comunes en la *Figura 2.4*.

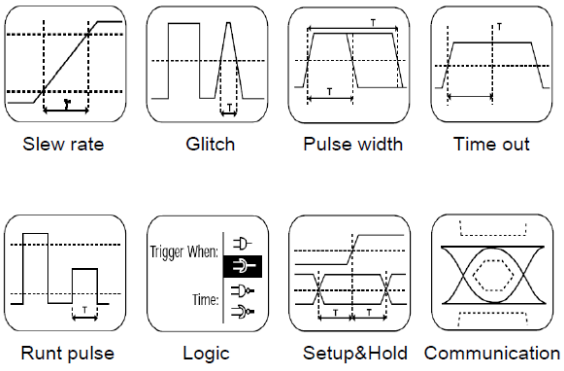



*Figura 2.4. Modos de disparo habituales*

Pero además de los modos, también podemos distinguir entre tres tipos de mecanismos de disparo, según la señal que se utilice para realizarlo. De esta manera, encontramos:

- **Disparo básico/interno (*INT*)**: Utiliza la propia señal que queremos visualizar como señal de referencia. Este es el caso más común y es el que utilizamos para poder mantener fija la imagen que se muestra en la pantalla. Será el tipo de *trigger* que utilizaremos en nuestro proyecto.

- **Disparo de línea/red eléctrica (*LINE*)**: Utiliza una fracción de la tensión de la red eléctrica como onda de referencia. Es muy útil cuando las señales que queremos visualizar están sincronizadas con un evento que viene marcado por la propia red de alimentación.

- **Disparo externo (*EXT*)**: Es utilizado en algunos osciloscopios que tienen un conector especial que permite introducir por él una señal específica de disparo, que será utilizada como onda de referencia.

### 2.1.4. Sondas

Las sondas realizan la función de conectar físicamente el elemento que queremos medir y el osciloscopio. Al realizar esta conexión se debe asegurar que el efecto de carga es mínimo para que así no se distorsione la señal.

Y es que, según el tipo de señal que queramos medir, se debe seleccionar un tipo de sonda diferente. Existen fundamentalmente dos tipos de sondas: pasivas y activas.

- **Sondas pasivas**

En este tipo de sondas la impedancia que presenta el osciloscopio en conjunto con la sonda no es constante con la frecuencia, haciendo que a frecuencias altas esta impedancia se reduzca notablemente, con el consiguiente aumento del efecto de carga.

Dentro de este tipo de sondas se encuentran las sondas divisoras de tensión, las cuales presentan una mayor resistencia de entrada y una menor capacidad en paralelo que el osciloscopio por sí solo, a costa de tener una atenuación constante de la señal. Suelen incluir una red RC con una resistencia de 9MΩ y un condensador ajustable, de forma que, cuando la sonda está compensada ($RC = R_e \cdot (C_e + C_c)$), la tensión que ve la entrada del osciloscopio está atenuada en un factor 10 (10X). Se debe por lo tanto considerar esta atenuación al realizar medidas de amplitud, aunque existen sondas que se pueden configurar para trabajar sin esa atenuación (1X).

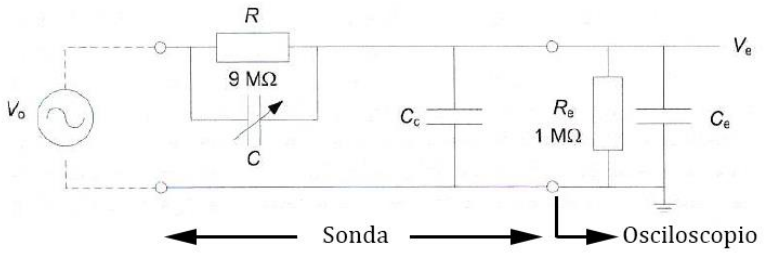



*Figura 2.5. Relación de impedancias de entrada (Sonda + Osciloscopio)*

- **Sondas activas**

Este tipo de sondas incorporan además elementos activos como transistores o amplificadores, los cuales requieren alimentación. Gracias a estas incorporaciones, permiten ofrecer impedancias elevadas a altas frecuencias sin perder resolución.

Suelen utilizarse para medir altas frecuencias, cuando la frecuencia de la señal de interés es superior a los 250 MHz. Este rango de frecuencias de entrada más elevado supone una ventaja, que solo se ve reducida por el rango de tensiones de entrada que admiten, mucho menor que el de las sondas pasivas.

Para nuestro caso, el aspecto más importante en el que debemos fijarnos es en que tengamos una buena adaptación de impedancias en el circuito de entrada del osciloscopio, por lo que habrá que escoger los componentes de manera oportuna.

Dado que las frecuencias de entrada admitidas por nuestro sistema estarán en la escala de los KHz, se usarán sondas pasivas para la realización de las medidas.

### 2.1.5. Interconexión

La etapa final en los osciloscopios consiste en la interpretación de las muestras tomadas para su representación por pantalla y poder observar la señal que hemos capturado. Habitualmente esta pantalla forma parte del propio componente, por lo que el envío de la información a la pantalla se realiza de forma interna. Pero en nuestro caso, la visualización se realizará a través de un dispositivo Android que estará conectado al resto del sistema mediante USB, por lo que el último aspecto a tener en cuenta tiene que ver con la interconexión de los dos subsistemas (*PCB* y dispositivo *Android*).

USB (*Universal Serial Bus*) es un estándar industrial que define los cables, conectores y protocolos usados en un determinado bus para conectar, comunicar y proveer de alimentación eléctrica ordenadores, periféricos y dispositivos electrónicos. Nació para unificar todos los conectores creando uno más sencillo y de mayores prestaciones.

A partir de 1995, el estándar USB se ha desarrollado para la conexión de una amplia gama de dispositivos y ha ido evolucionando desde entonces, añadiendo nuevos modos y diferentes velocidades. La especificación actual es el estándar USB 3.0.

La especificación USB define tres velocidades diferentes, pensadas para distintos tipos de dispositivos. Son velocidades del sistema y no representan el rendimiento del mismo, el cual siempre será menor que esas velocidades.

| Nombre | Velocidad |
|---|---|
| **Low Speed** | 1,5 Mbit/s |
| **Full Speed** | 12 Mbit/s |
| **High Speed** | 480 Mbit/s |

*Tabla 2.1. Relación de velocidades en USB*

USB implementa una topología de estrella en niveles (con un máximo de 127 dispositivos conectados) y se basa en el paso de un testigo, el cual distribuye el *host*.

El host es el maestro, y es quien inicia todas las comunicaciones, de forma que no puede haber comunicaciones entre dispositivos.

El dispositivo cuya dirección coincide con la que tiene el testigo responde aceptando o enviando datos al *host*. Este también gestiona la distribución de energía a los periféricos que lo requieran, donde los dispositivos solo pueden consumir 100 mA o hasta 500 mA si el *host* lo permite.

Los cables están pensados para que no se puedan realizar de manera incorrecta las interconexiones entre subsistemas, de manera que no se puedan conectar juntos dos *host* o dos dispositivos. USB requiere de un cable apantallado con cuatro hilos conductores: dos para la alimentación (tierra y $V_{bus}$, el cual lleva 5 V) y otro par (D+ y D-) responsable de la comunicación diferencial de datos.

En lo que respecta a los conectores, USB emplea dos tipos para evitar de nuevo conexiones incorrectas: serie A y serie B. Los primeros apuntan desde el *host* hacia el dispositivo, mientras que los segundos lo hacen de manera contraria. Existen además distintos receptáculos que se presentan diferentes variantes y que difieren en el número de pines, la forma y el tamaño.

Existen cuatro formas diferentes de transmitir datos en un bus USB. Cada una corresponde a un tipo de transferencia (flujo de datos) y cada una tiene su propósito y características, que determinan parámetros como la frecuencia, la longitud de la transacción o la verificación mediante un CRC (*Cyclical Redundancy Checksum*).

| | ***Control*** | ***Bulk*** | ***Interrupt*** | ***Isochronous*** |
|---|---|---|---|---|
| *Tam. Max. datos (Low Speed)* | 8 bytes | No soportado | 8 bytes | No soportado |
| *Tam. Max. datos (Full Speed)* | 64 bytes | 64 bytes | 64 bytes | 1023 bytes |
| *Tam. Max. datos (High Speed)* | 64 bytes | 512 bytes | 1024 bytes | 1024 bytes |
| *CRC* | Si | Si | Si | No |
| *Ancho de banda garantizado* | No | No | No | Si |
| *Garantiza periodicidad* | No | No | Si | Si |
| *Usos frecuentes* | Configuración | Impresoras, escáner | Teclados, ratones | Audio, video |

*Tabla 2.2. Tipos de transferencias USB y características*

Para seleccionar el tipo de transferencia que debemos utilizar debemos tener en cuenta ciertas consideraciones: latencia, comprobación de errores, ancho de banda, control y configuración.

En cuanto al funcionamiento interno del protocolo de control, configuración y comunicación, así como el formato de los mensajes intercambiados, no se desarrollará información debido a que es un tema bastante extenso y que no debemos analizar para tomar las consideraciones de diseño y fabricación. Solamente será necesario conocer esta información a la hora de realizar la programación tanto de Hardware como de Software.

Para el caso de estudio en este proyecto, la *PCB* hará las veces de *host*, iniciando las comunicaciones y tomando las decisiones oportunas, por lo que necesitará un conector de tipo A, y el dispositivo *Android* desempeñará el papel de dispositivo USB, aprovechando el conector de tipo B que incluyen por defecto. Trabajaremos en el modo *Full Speed* y utilizaremos transferencias isócronas, las cuales suelen utilizarse para la transferencia de flujos de datos de audio y video por tener el ancho de banda garantizado, a costa de suprimir la comprobación de errores.

## 2.2. Sistema: descripción por bloques

Para entender de mejor manera el funcionamiento de los osciloscopios digitales analizaremos qué le pasa a la señal que queremos medir desde que conectamos la sonda al circuito, hasta que obtenemos los resultados en pantalla. Nos ayudaremos del esquema de bloques mostrado en la *Figura 2.6*, el cual iremos relacionando con las partes de nuestro proyecto.

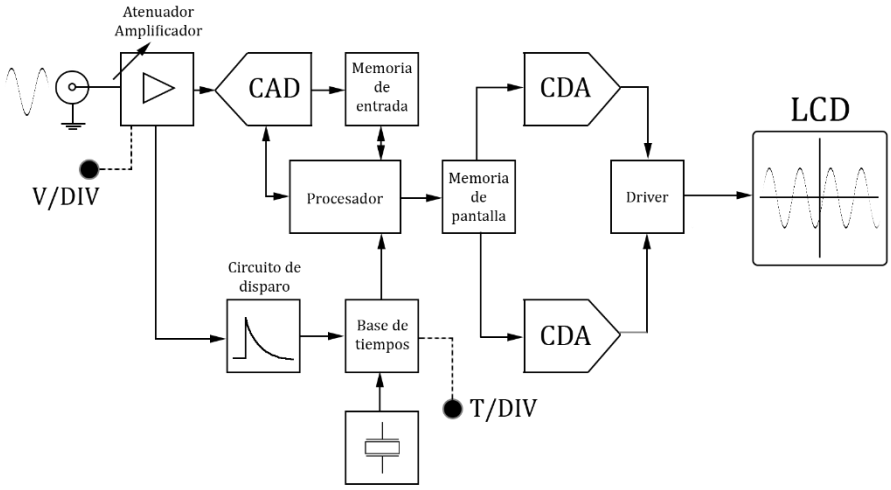



*Figura 2.6. Diagrama de bloques de un DSO*

Cuando conectamos la sonda al sistema que se quiere medir, la señal de entrada la atraviesa y se dirige a lo que conocemos como sección vertical, compuesta por el circuito atenuador y amplificador. Antes de este circuito debemos asegurarnos, como se comentó en la sección dedicada a las sondas, que la adaptación de impedancias es adecuada. Dependiendo de donde situemos el control del amplificador vertical atenuaremos o amplificaremos la señal, lo que nos permitirá medir señales más grandes sin que saturen la pantalla y señales más pequeñas con un mayor detalle. Además, se deberá ajustar el voltaje de la señal al rango de voltajes de entrada del conversor.

Una vez adecuada la señal a la entrada del CAD, este último muestrea la señal a intervalos de tiempo determinados (que marca la base de tiempo que seleccionemos en la sección horizontal) y convierte la señal de voltaje continua en una sucesión de valores digitales (muestras), que son almacenados en la memoria de entrada. Todo este proceso es controlado y administrado por un procesador.

Finalmente, y tras procesar los datos muestreados si es necesario, la sección de visualización recibe estos puntos almacenados en la memoria para presentar en pantalla la señal, de forma que se lleva a cabo la acción inversa a la primera conversión, pudiendo visualizar la señal gracias al driver correspondiente.

A pesar de que el funcionamiento es prácticamente el mismo, en nuestro proyecto existen ciertas diferencias con respecto al diagrama de bloques descrito anteriormente:

- Mientras que los selectores de escala vertical y horizontal suelen ser controles independientes que el usuario manipula por medio de botones físicos o selectores rotatorios, el osciloscopio diseñado solo mantendrá esos selectores físicos para el escalado vertical, mientras que para el horizontal se realizará desde el dispositivo *Android*, por medio de la pantalla táctil.

- En este ejemplo de diagrama, vemos que el CAD es un elemento independiente, controlado por el procesador y que las muestras obtenidas se almacenan en una memoria externa, mientras que en nuestro proyecto el procesador *DSP* ya tiene integrado el CAD y los datos se almacenan en la memoria RAM del procesador hasta su posterior envío.

- En nuestro caso, como dijimos anteriormente, los datos muestreados se enviarán por USB al dispositivo, quien será el encargado de interpretar esos datos para poder dar una visualización en pantalla de la señal, por lo que el sistema no está integrado como en el DSO descrito, donde la pantalla LCD forma parte del sistema conjunto.

El diagrama de bloques que describe por tanto nuestro sistema vendría a ser el siguiente.

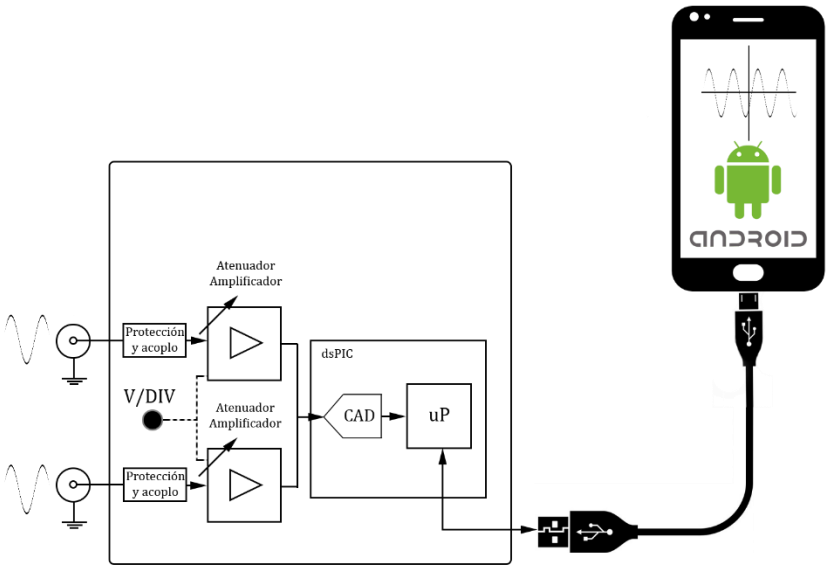



*Figura 2.7. Diagrama de bloques del osciloscopio del proyecto*

# Capítulo 3. Hardware

En este capítulo se analizará en detalle el Hardware del sistema (es decir, la *PCB*) desde la etapa de diseño, explicando cada etapa del circuito que la compone, así como sus componentes y cálculos necesarios, hasta la etapa de fabricación y su posterior montaje.

Para las labores de diseño, tanto del esquemático del circuito, como del *layout* del mismo se utilizó la Suite de Proteus en su versión Proteus 7 Professional.

## 3.1. Análisis por etapas

Al analizar cada etapa por separado, se hará referencia únicamente a un canal, ya que las explicaciones para el segundo canal serán análogas. Además, las capturas del esquemático del circuito que se utilizarán como apoyo para la descripción son solo extractos de los diagramas de circuito del sistema completo, los cuales pueden encontrarse en el Anexo B.

Como último detalle los componentes destacados en negrita fueron reemplazados o sustituidos por otros equivalentes. En los apartados correspondientes se justifica de manera breve su sustitución.

### 3.1.1. Circuito de alimentación

A pesar de que no es considerado como una etapa del circuito debido a que no influye en el camino de la señal de entrada, a continuación, se describe brevemente el circuito de alimentación del circuito.

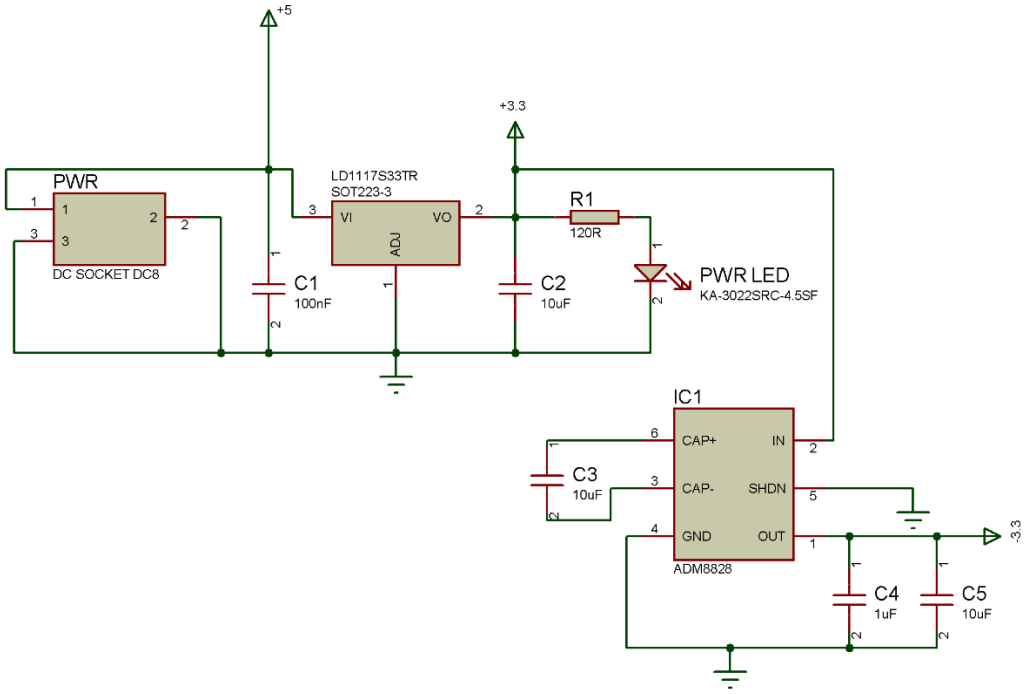



*Figura 3.1. Diagrama del circuito de alimentación*

Como vemos en la *Figura 3.1*, este circuito está formado por dos elementos principales que se encargan de convertir el voltaje de entrada, para adecuarlo a la alimentación demandada por los componentes activos del circuito. En concreto consta de un regulador de voltaje *LD1117S33TR* que transforma los 5 V de entrada a 3,3 V, necesarios para la alimentación del DSP y la alimentación positiva de los amplificadores operacionales, y un inversor de voltaje *ADM8828* que se encarga de obtener (a partir de los 3,3 V del regulador) la tensión de -3,3 V necesaria para la alimentación negativa de los amplificadores operacionales.

Además, podemos ver el conector de entrada, de tipo *jack*, a cuya salida irá conectada la línea de alimentación de 5 V (para el USB), y el diodo LED que indicará el encendido del sistema.

### 3.1.2. Circuito de entrada

La etapa inicial del circuito corresponde al circuito de entrada del osciloscopio y está formada por varias partes, las cuales se muestran en la *Figura 3.2.* Cada una de las ellas tiene una función determinada.

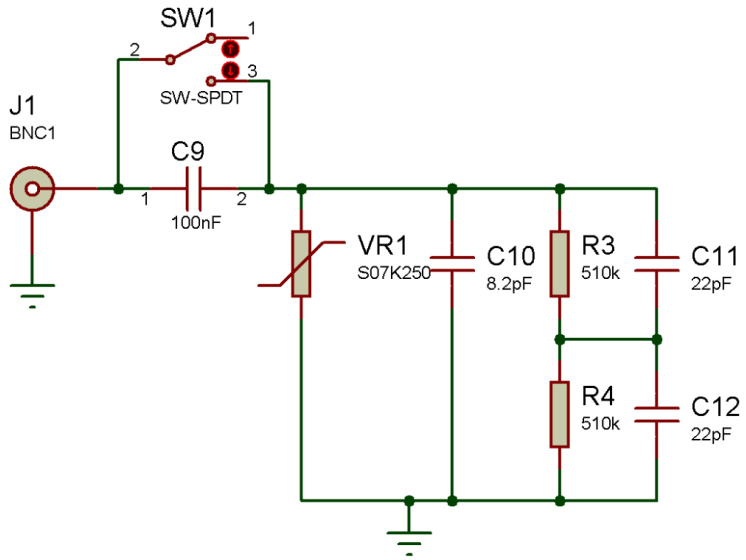



*Figura 3.2. Diagrama del circuito de entrada*

A la entrada del circuito encontramos un conector BNC, que permite la conexión de las sondas para medir las señales.

A continuación, al igual que en los osciloscopios comerciales, tenemos la opción de seleccionar el tipo de acoplamiento para nuestra entrada, pudiendo elegir si la señal pasará directamente (acoplo DC), o lo hará a través del condensador correspondiente (acoplo AC). Para accionar el tipo de acoplamiento, se colocó un interruptor SPDT (un polo y dos vías).

Como el circuito debe ser capaz de medir señales con un voltaje elevado, se vuelve imprescindible instalar una protección contra sobrevoltajes, evitando de esta manera cualquier daño en los elementos e integrados que conforman las etapas

posteriores. Para ello se instaló un varistor de 250 V. La corriente elevada, por otra parte, no está controlada, aunque hubiese bastado con colocar un fusible a continuación del conector BNC. Se asume que al estar enfocado a realizar prácticas en laboratorios docentes la corriente suministrada por los equipos generadores estará limitada. No obstante, sería interesante que en versiones posteriores del Hardware se instalase este elemento para asegurar de esta manera la protección mencionada anteriormente.

Cuando estamos midiendo señales eléctricas debemos asegurarnos de que no vamos a alterar la señal de interés introduciendo cargas adicionales a través del sistema de medida. Para ello debemos obtener un buen acoplo de impedancias a la entrada, haciendo que la resistencia de entrada del osciloscopio sea elevada. Habitualmente, las sondas utilizadas en osciloscopios tienen una resistencia de 9 MΩ y los osciloscopios una impedancia de entrada de 1 MΩ en paralelo con una capacidad de unos 20 pF. En nuestro circuito esta asociación la logramos mediante una serie de dos resistencias de 510 KΩ (que conforman esa resistencia de 1MΩ), que sirven como divisor de voltaje (dividiendo a la mitad la amplitud de la señal medida), permitiéndonos medir señales de mayor amplitud, en paralelo con una red de condensadores que configuran una capacitancia de entrada de 19,2 pF, cercana a los 20 pF comentados antes.

Despreciamos las impedancias del amplificador operacional de la etapa siguiente, ya que su resistencia de entrada es superior a los 100 MΩ y su capacidad cercana a los 0,5 pF.

### 3.1.3. Amplificador/Atenuador

El circuito amplificador/atenuador establece la amplitud de la señal de interés a la entrada del CAD. De este modo protegemos al sistema, a la vez que nos posibilita apreciar la señal de manera adecuada, aprovechando al máximo el rango de voltaje de entrada del CAD.

El circuito mostrado en la *Figura 3.3* está formado por un amplificador operacional en configuración inversora (entrada por el terminal negativo) con ganancia ajustable a través de una red de resistencias que seleccionamos por medio de un interruptor de tipo DIP con 4 selectores.

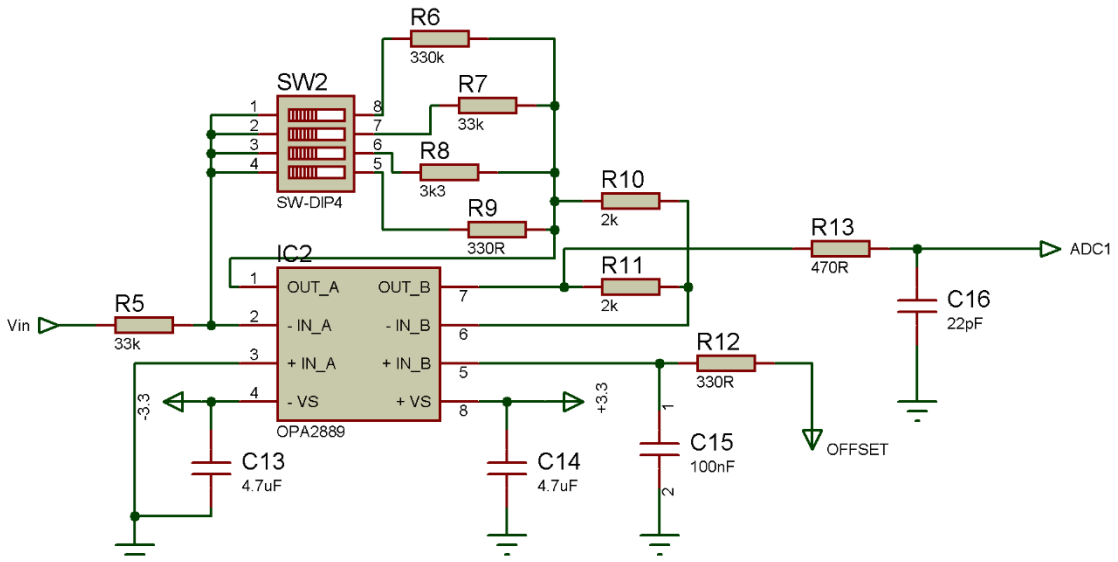



*Figura 3.3. Circuito amplificador/atenuador y acondicionamiento*

La ecuación que determina la función de transferencia del amplificador, y por tanto la ganancia es la siguiente.

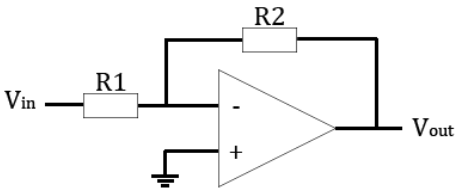



*Figura 3.4. Esquema de un amplificador inversor*

$$V_{OUT} = -V_{IN} \cdot \frac{R_2}{R_1}$$

*Ecuación 3.1. Función de transferencia del amplificador operacional*

En nuestro caso, $R_1$ tiene un valor de 33 KΩ ($R_5$ en *Figura 3.3*), mientras que $R_2$ se selecciona de la red de resistencias que van escaladas por décadas, desde 330 Ω, hasta 330 KΩ ($R_6$ a $R_9$ en *Figura 3.3*), lo cual nos da unas ganancias desde x10 hasta x0,01 respectivamente. En nuestro caso no se añadieron condensadores de compensación, aunque sería recomendable para que no apareciesen efectos de carga variables con la frecuencia. Si se añaden condensadores en paralelo con $R_5$ y $R_{6\text{-}9}$ de manera que $C_{6\text{-}9}R_{6\text{-}9} = C_5R_5$, la tensión de entrada al amplificador no dependerá de la frecuencia.

Como las frecuencias de entrada de las señales estarían en la escala de los KHz, necesitamos que el amplificador operacional tenga un ancho de banda y respuesta en frecuencia mayor. Inicialmente fueron seleccionados los **OPA2889** de Texas Instruments, con un ancho de banda de unos 60 MHz, pero por falta de existencias en el almacén del proveedor, finalmente se sustituyeron por los **OPA2890**, cuya frecuencia límite está en torno a los 90 MHz. Las características son prácticamente idénticas, aunque estos últimos consumen algo más de corriente cuando están activos.

### 3.1.4. Acondicionamiento de voltaje (*Offset*)

Una vez la señal ha sido amplificada o atenuada, la siguiente etapa se encarga de invertir de nuevo la señal, haciéndola pasar por otro amplificador operacional en configuración inversora, de forma que tengamos la señal original solamente escalada. En este caso, el par de resistencias que configuran la ganancia de esta etapa inversora, se eligieron con el mismo valor, de forma que esta ganancia fuese unitaria y no modificásemos la amplitud de la señal. A pesar de que originalmente las resistencias propuestas para el par de ganancia unitaria eran de **2 KΩ**, fueron finalmente sustituidas por dos resistencias de **33 KΩ**, por no tener stock en ese momento de las primeras.

Este paso podemos verlo en la *Figura 3.3* (se usa el segundo amplificador operacional del OPA2889), donde además vemos que, a la entrada positiva del este segundo operacional se le aplica una tensión de offset que sirve para sumar una tensión media a la señal de entrada de forma que se adapte de una mejor manera a la entrada del CAD.

Para determinar la función de transferencia de este segundo amplificador debemos analizar el circuito por superposición para así ver la contribución del voltaje de *offset* a la salida.

Para el cálculo del efecto de $V_{in}$ a la entrada el esquema del circuito amplificador sería el mismo que el de la *Figura 3.4*.

$$V_{OUT,1} = -V_{IN} \cdot \frac{R_2}{R_1} \xrightarrow{R_1=R_2} = -V_{IN}$$

*Ecuación 3.2. Cálculo de la contribución de $V_{IN}$*

Ahora se fija $V_{in}$ en 0 V y se determina la contribución del voltaje de *offset* a la salida. Podemos ver el esquema de circuito en la *Figura 3.5*.

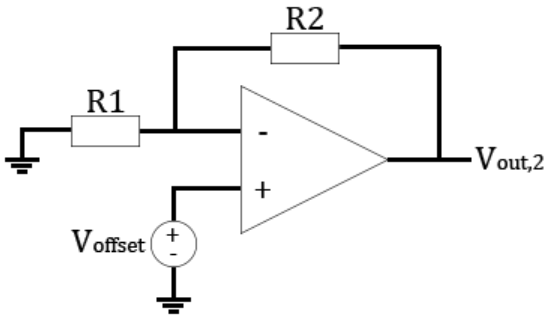



*Figura 3.5. Contribucion de $V_{offset}$ en el segundo amplificador*

$$V_{OUT,2} = \left(1 + \frac{R_2}{R_1}\right) \cdot V_{OFFSET} \xrightarrow{R_1=R_2} = 2 \cdot V_{OFFSET}$$

*Ecuación 3.3. Cálculo de la contribución de $V_{Offset}$*

Sumando ambas contribuciones obtenemos la función de transferencia total del segundo amplificador operacional, cuyo esquema final podemos ver en la *Figura 3.6*.

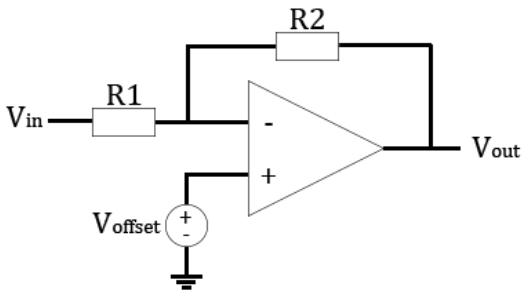



*Figura 3.6. Esquema de circuito final del segundo amplificador*

$$V_{OUT} = V_{OUT,1} + V_{OUT,2} = -V_{IN} + 2 \cdot V_{OFFSET}$$

*Ecuación 3.4. Función de transferencia del segundo amplificador operacional*

Una vez realizado el cálculo, comprobamos cuál es el rango de entrada al CAD del microcontrolador elegido. Para el caso del dsPIC escogido este rango de voltaje es de 0-3,3V, por lo que para adaptar lo mejor posible las señales de entrada debemos sumarle una tensión de 1,65 V a la señal. A la vista de los cálculos, $V_{Offset}$ deberá valer 0,825 V.

Para conseguir este valor de voltaje de la forma más precisa posible se pensó en realizar un divisor de tensión con una resistencia variable (**potenciómetro**) como se puede ver en la *Figura 3.7*, pero tras colocar el componente en el *layout* definitivo no fue posible conseguirlo (por falta de stock del componente), por lo que finalmente fue sustituido por una resistencia de valor lo más ajustado posible para conseguir ese valor de voltaje.

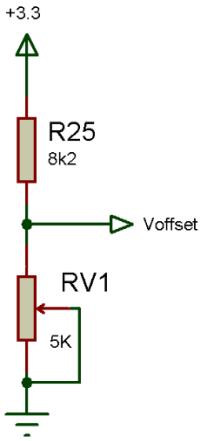



*Figura 3.7. Divisor de voltaje para obtener $V_{Offset}$*

El valor escogido fue de **2,7 KΩ**, consiguiendo un valor de $V_{Offset}$ de 0,817 V, como se ve en el siguiente cálculo.

$$V_{Offset} = 3{,}3\ [\mathrm{V}] \cdot \frac{2{,}7\ [\mathrm{K\Omega}]}{(8{,}2 + 2{,}7)[\mathrm{K\Omega}]} = 0{,}817\ [V]$$

*Ecuación 3.5. Cálculo de $V_{Offset}$ final*

Una vez determinadas las diferentes amplificaciones, y conociendo el valor del rango de voltaje de entrada del CAD, podemos definir cuáles serán los niveles de voltaje asignados a cada ganancia, o lo que es lo mismo, a cada posición del interruptor.

| Posición (Ganancia) | Voltaje mín. | Voltaje máx. |
|---|---|---|
| **1 (x10)** | -0,165 V | 0,165 V |
| **2 (x1)** | -1,65 V | 1,65 V |
| **3 (x0,1)** | -16,5 V | 16,5 V |
| **4 (x0,01)** | -165 V | 165 V |

*Tabla 3.1. Relación de ganancias y niveles de voltaje de entrada*

Por último, al final de esta etapa encontramos un filtro RC formado por una resistencia de 470 Ω ($R_{13}$ en *Figura 3.3*) y un condensador de 22 pF ($C_{16}$ en *Figura 3.3*), los cuales actúan a modo de filtro paso bajo, reduciendo posibles sobre picos en la señal a la entrada del CAD tras las amplificaciones. La frecuencia de corte de este filtro es de unos 15 MHz.

$$f_c = \frac{1}{2\pi R_{13} C_{16}} = \frac{1}{2\pi \cdot 470[\Omega] \cdot 22[pF]} = 15{,}3999\ [MHz]$$

Aunque el filtro no es del todo necesario, la resistencia $R_{13}$ actúa como limitador de corriente a la entrada del CAD, por lo que es recomendable dejarla.

### 3.1.5. dsPIC

Como dijimos anteriormente para la captura y procesamiento de las señales de entrada elegimos un *DSP* de la empresa *Microchip Technology*. En concreto el modelo dsPIC33EP256MU806. También será el encargado de controlar las comunicaciones con el dispositivo *Android*. Las características principales de este microcontrolador que más nos conciernen para nuestro proyecto se resumen a continuación.

- Arquitectura Harvard Modificada.
- Datos de 16 bits.
- Instrucciones de 24 bits.
- Velocidad máxima de CPU de 70 MIPS.
- 256 KB de memoria Flash (memoria de programa).
- 28 KB de memoria SRAM (memoria de datos).
- Interfaz USB 2.0 con soporte *Full Speed* y OTG.
- 2 conversores analógico digitales independientes.
- 7 fuentes de trampa no enmascarables y hasta 114 fuentes de interrupción.

A pesar de que el DSP cuenta con multitud de fuentes de interrupción, en nuestro caso únicamente haremos uso de algunas de ellas. En concreto utilizaremos dos interrupciones internas correspondientes a los dos CADs una vez terminan una conversión, y una externa (INT0) utilizada en el caso de que queramos que la señal del canal 2 sea la fuente de disparo para la señal del canal 1.

Tras las etapas anteriores en las que adaptábamos y acondicionábamos la señal de entrada, necesitamos de un conversor analógico digital que muestree la señal y convierta sus valores. Debido al rango de valores de entrada que estamos manejando, sobre todo por el nivel inferior (la escala de voltajes más pequeña), se hace necesario disponer de una resolución de al menos 10 bits.

Además, necesitamos que el conversor tenga una alta velocidad de muestreo y conversión, de como mínimo 500 Ksps (500000 muestras por segundo) para poder muestrear señales, según el teorema de Nyquist, de hasta 250 KHz, lo cual es un rango aceptable para nuestro proyecto.

En concreto el dsPIC elegido posee dos CADs independientes con resoluciones de hasta 12 bits en el primero y hasta 10 bits en el segundo, y una tasa de muestreo de hasta 1,1 Msps trabajando con 10 bits. Cada uno de ellos con 4 *S&H* (*Sample and Hold*), o circuitos de muestreo y retención, y 24 canales de entrada.

La parte del circuito que comprende al dsPIC y sus conexiones puede verse en la *Figura 3.8*, aunque para una mayor comodidad es preferible referirse al Anexo B, donde se muestra una versión más ampliada.

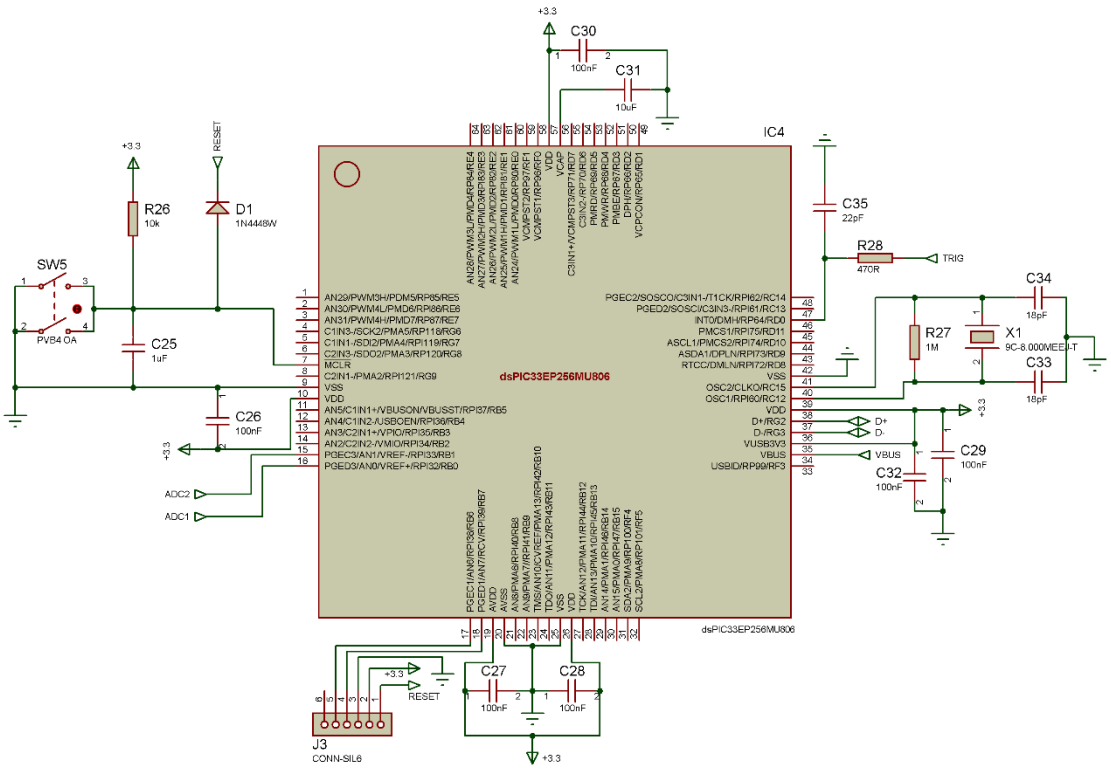



*Figura 3.8. Circuito de conexionado del dsPIC*

Cuando se trabaja con este tipo de microcontroladores es indispensable realizar un mínimo de conexiones entre algunos de los pines del mismo antes de comenzar con el desarrollo. Esta configuración corresponde a los siguientes pines.

- Todos los pines de $V_{DD}$, $V_{SS}$, $AV_{DD}$ y $AV_{SS}$ deben usar condensadores de desacoplo entre pares de los mismos. Estos condensadores deben ser de unos 100 nF y deben estar colocados lo más cerca posible de los pines a los que están conectados.

- En el caso del pin $V_{CAP}$ se debe utilizar otro condensador de 10 µF para estabilizar el regulador de voltaje de salida.

- En nuestro caso, como también hacemos uso del periférico USB, el pin $V_{USB3V3}$ debe estar conectado a la fuente de 3,3 V.

- El pin de *reset* rotulado como $\overline{MCLR}$ debe conectarse mediante un pulsador que permita reiniciar el programa, pero también debe conectarse a la línea de *reset* del programador de manera que este pueda llevar la línea a 0 V durante las tareas de programación o depuración del dispositivo.

- Los pines PGECx y PGEDx deben estar conectados a las líneas correspondientes del programador, para las tareas de programación y depuración del dispositivo. Estas líneas deben mantenerse lo más cortas posibles. Además, no se recomienda colocar resistencias de *pull-up*, condensadores o diodos en serie en estas líneas para que no interfieran con la comunicación.

- Por último, los pines correspondientes al oscilador deben estar conectados al mismo, en la misma cara de la *PCB* y de nuevo lo más cerca posible de los pines del *DSP*. Para seleccionar la frecuencia más adecuada en cada caso, debemos fijarnos, no solo en la frecuencia (velocidad) a la que queremos hacer trabajar a nuestro procesador, sino también en la frecuencia que necesitan los periféricos que utilizamos para funcionar correctamente. En nuestro caso, utilizamos el periférico USB, el cual necesita una frecuencia de 48 MHz para funcionar. Esta frecuencia se consigue configurando internamente un PLL auxiliar del propio dsPIC.

A la hora del montaje, no se pudo disponer de un **cristal de cuarzo** con la frecuencia óptima para poder conseguir esa frecuencia de manera exacta, por lo que se utilizó uno de frecuencia algo superior (**14,7456 MHz**) con el que se consiguió una frecuencia de funcionamiento para USB de 49 MHz.

Para más información referente al dsPIC y sus componentes se puede consultar la hoja de especificaciones incluida en la bibliografía del trabajo (nº 21).

## 3.2. Fabricación

Una vez finalizada la labor de diseño del circuito y escogidos los componentes del mismo, se pasó a la parte de diseño del *layout* de la *PCB*.

Primero se configuraron los parámetros característicos de la placa de circuito (reglas de diseño) tales como el número de capas, el tamaño, el grosor, el espacio entre pistas, etc., siguiendo las directrices que nos proporcionó la empresa de fabricación (ver Anexo D).

Una vez establecidos los parámetros definimos los encapsulados de los componentes a utilizar y comprobamos si su huella estaba ya diseñada en ARES (la parte de Proteus para este tipo de diseños) o debíamos diseñarla nosotros. Esto último solo fue necesario en algunos componentes, como los conectores BNC, la ferrita, el conector de alimentación y el USB, el diodo LED y el pulsador. Para diseñarlas, partimos de las hojas de especificaciones de los componentes. Además, fijamos la serigrafía de los componentes, así como algunas marcas de posición de manera que fuese más sencillo identificarlos y situarlos en la *PCB*.

A continuación, se colocaron los componentes en la placa de manera que creásemos un esquema limpio y ordenado, que favoreciese un rutado de pistas cómodo y sin muchos saltos entre capas para mejorar así la inmunidad ante interferencias y reforzar su compatibilidad electromagnética.

Como decíamos, luego se procedió al rutado de pistas y a la colocación de los planos de masa correspondientes. Después se añadieron algunas identificaciones en la máscara de componentes. El resultado final puede verse en la *Figura 3.9*.

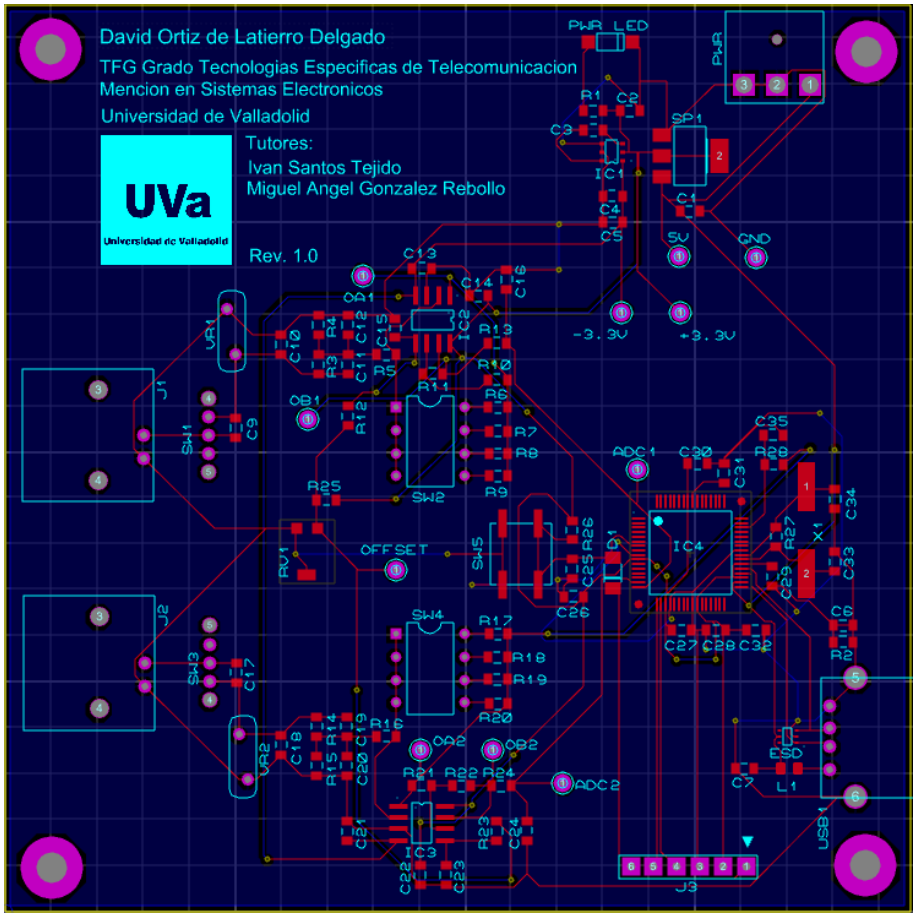



*Figura 3.9. Layout definitivo de la PCB*

Finalmente, y tras realizar un test de preproducción para comprobar que no habíamos incumplido ninguna regla de diseño, se generaron los ficheros para fabricación, los cuales se enviaron al fabricante (*PCBWay*).

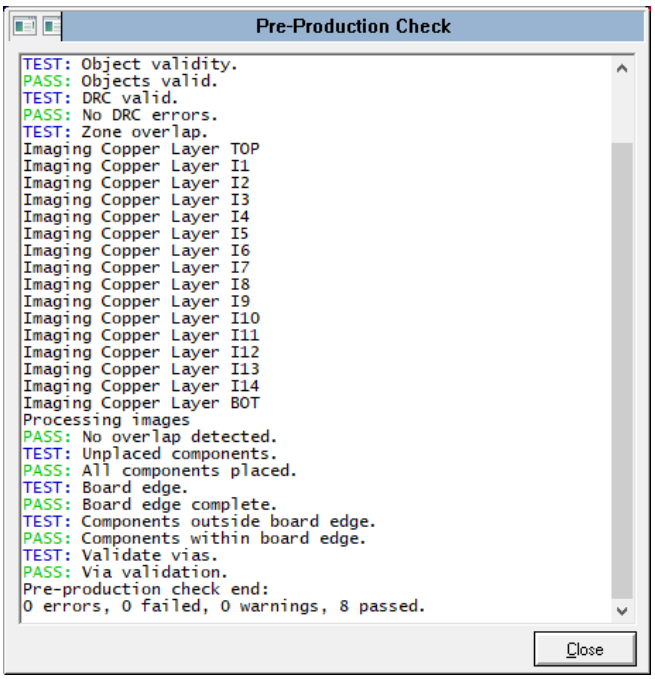



*Figura 3.10. Test de preproducción superado*

A continuación, podemos ver el aspecto de la *PCB* producida una vez llegó de la empresa fabricante. Podemos apreciar el gran acabado, tanto en la terminación de los contactos para soldar los componentes como en la definición de la máscara de componentes.

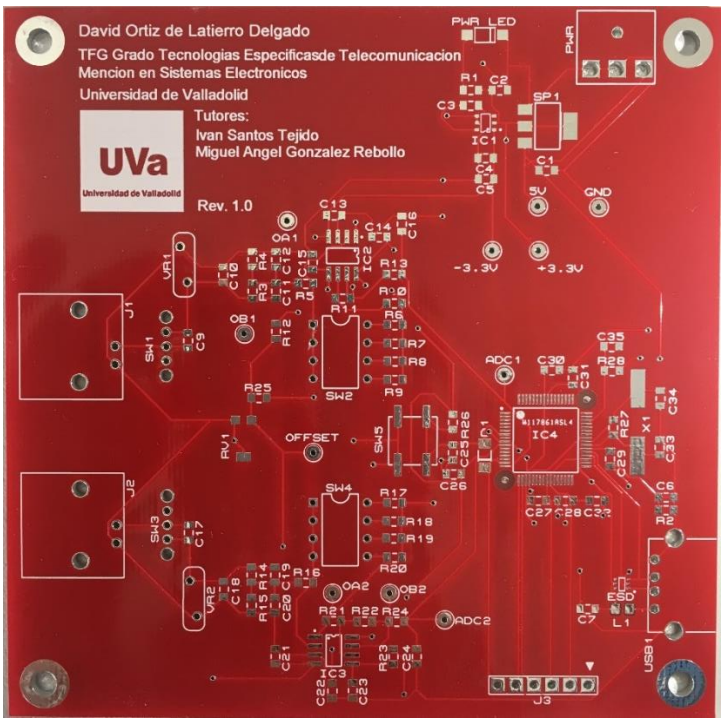



*Figura 3.11. Foto real de la PCB recién fabricada*

## 3.3. Montaje

Antes de la realización del montaje, se comprobó la disponibilidad de material en el laboratorio, anotando los componentes que ya había en stock para solo hacer el pedido de los necesarios. Como ya hemos comentado en puntos anteriores, algunos de los componentes que era preciso pedir no estaban disponibles en la web del proveedor o estaban fuera de inventario, por lo que finalmente fueron reemplazados por otro similar o de un valor asumible.

Una vez llegaron los componentes comenzó el proceso de soldado, para lo cual fue indispensable contar con ayuda de los equipos disponibles en el laboratorio (soldadores de punta fina, microscopios ópticos y microscopios digitales conectados a un monitor LCD), ya que la mayor parte de los componentes son de montaje superficial (*SMC*) y sus encapsulados muy pequeños. Alguno de ellos, como el dsPIC o el inversor de voltaje para conseguir -3,3 V, hubo que comprobarlos en varias ocasiones debido a un mal contacto con los *pads* de la placa. Podemos ver el montaje de los componentes en la placa en la *Figura 3.12*.

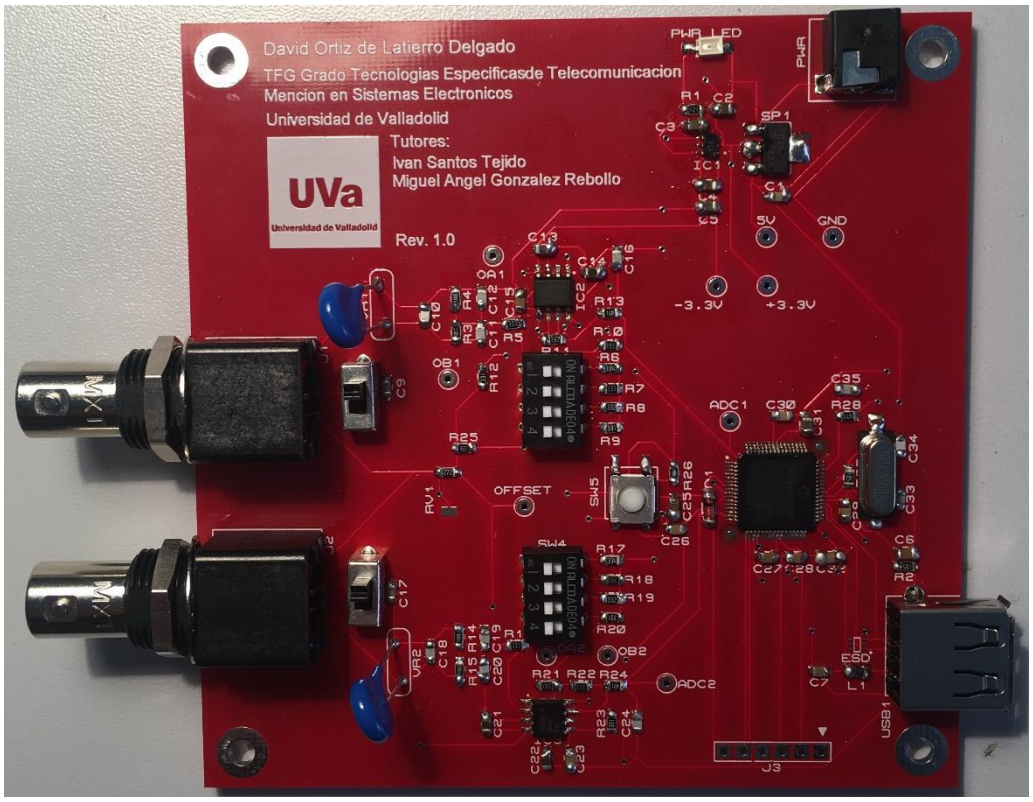



*Figura 3.12. Imagen de la PCB con los componentes montados*

Además de las dificultades descritas anteriormente hubo un problema con el integrado protector frente a descargas electrostáticas (***ESD***), cuya huella fue mal importada en el Software de diseño y no coincidía con el componente real, por lo que se suprimió su colocación. Es recomendable que en posteriores versiones de la *PCB* se rediseñe el encapsulado del componente y se coloque en el circuito, para proteger así frente a este tipo de descargas a los componentes más críticos del circuito.

Tras completar el soldado de todos los componentes, procedimos a alimentar la placa para comprobar que todo funcionaba correctamente y los reguladores de voltaje proporcionaban la salida deseada. Al medir con un multímetro los puntos de prueba de la placa se comprobó que existía un mal funcionamiento de la misma ya que los voltajes de 5 V y 0 V (*GND*, tierra) estaban cortocircuitados a la entrada del circuito. Se comprobó que esto era debido a que la huella del conector de alimentación tenía sus pines invertidos (de nuevo ocasionado por un mal diseño del encapsulado del componente). La solución más simple fue desmontar el componente y volver a soldarlo por la parte de debajo de la *PCB*, de forma que los pines estuviesen en la posición correcta.

Después de solucionar el problema se comprobó que todos los voltajes del circuito eran correctos y se procedió a comprobar si el camino de la señal de entrada estaba bien definido y la señal a lo largo del circuito se adaptaba y acondicionaba como debiera. Para ello se utilizó un generador de funciones con una onda sinusoidal que era llevada a la entrada del circuito conectando una sonda al conector BNC. Estas medidas también fueron satisfactorias.

Finalmente se conectó la placa al ordenador mediante el Hardware programador del propio fabricante del *DSP*, para asegurarnos de que el ordenador reconocía la placa y podíamos cargar un programa de prueba en el dsPIC.

Tras garantizar que todas las pruebas habían sido satisfactorias ya estábamos en condiciones para realizar el programa principal que cargaríamos en el microcontrolador y probar el modelo real, la placa que hemos fabricado y la que presentaremos como producto final del trabajo.

## 3.4. Consumo de potencia

Para la alimentación del circuito se requiere de una fuente de alimentación externa que suministre los niveles de 5 V y 0 V, ya que el resto de valores serán generados en el circuito de alimentación.

Para decidir el amperaje necesario de la fuente, se debe realizar un estudio de consumo para los diferentes componentes activos del circuito, es decir los que consumen energía. Podemos ver estos componentes en la *Tabla 3.2.*

En la siguiente tabla se incluyen los valores máximos de consumo por cada componente y el número de componentes de cada tipo.

| Elemento | Consumo de corriente por Ud. | Cantidad | Consumo total |
|---|---|---|---|
| **dsPIC33EP256MU806** | 93 mA | 1 | 93 mA |
| **ADM8828** | 1 mA | 1 | 1 mA |
| **OPA2890** | 75 mA | 2 | 150 mA |
| **Diodo LED** | 27,5 mA | 1 | 27,5 mA |
| | | **Total** | **271,5 mA** |

*Tabla 3.2. Consumo de corriente de los componentes activos*

Teniendo los valores totales máximos de corriente para los componentes, podemos observar que con una fuente de alimentación que proporcione 500 mA será más que suficiente.

Además, como todos los componentes activos están conectados a la alimentación de 3,3 V, podemos concluir que el consumo de potencia que tendrá la *PCB* de nuestro proyecto será de unos 900 mW.

$$P_{máx} = 3{,}3\ [V] \cdot 271{,}5\ [mA] = 895{,}95\ [mW]$$

*Ecuación 3.6. Calculo del consumo de potencia del circuito*

## 3.5. Lista de elementos afectados

Después de analizar cada etapa del circuito por separado y realizar el montaje final de los componentes en la PCB, encontramos que ciertos elementos sufrieron modificaciones o sustituciones de última hora. Como se indicó en la introducción del presente capítulo, todos ellos fueron resaltados en negrita.

La *Tabla 3.3* resume estos elementos.

| Componente | Sustitución | Motivo |
|---|---|---|
| OPA2889 | OPA2890 | Falta de stock en almacén proveedor |
| $R_{10\text{-}11}$, $R_{21\text{-}22}$ (2 KΩ) | Resistencias de 33 KΩ | Falta de stock en laboratorio |
| Potenciómetro (5 KΩ) | Resistencia de 2,7 KΩ | Falta de stock en almacén proveedor |
| Cristal 8 MHz | Cristal 14,7456 MHz | Falta de stock en laboratorio |
| ESD | N/A | Huella mal diseñada |
| Conector alimentación | N/A | Huella mal diseñada |

*Tabla 3.3. Listado de componentes modificados o sustituidos*

Mientras que la sustitución de los amplificadores operacionales o las resistencias de la etapa de offset de cada canal ($R_{10\text{-}11}$, $R_{21\text{-}22}$) pueden permanecer con su sustitutivo, no es el caso del resto de componentes.

Para el ESD y el conector de alimentación se debería rediseñar la huella de forma correcta en una posterior versión de la PCB.

Se debería conseguir el potenciómetro para ajustar el valor de la tensión de *offset* de la manera más óptima posible y, en caso de no disponer del componente exacto, sustituirlo por otro potenciómetro, diseñando su huella de nuevo.

El cristal también debería ser sustituido por otro de frecuencia 8 MHz para así poder asegurar que se consigue de manera correcta la frecuencia necesaria para el periférico USB y poder hacer trabajar con los conversores a la máxima tasa de muestreo.

# Capítulo 4. Programación del Hardware (Firmware)

Como ya hemos comentado, para este proyecto se eligió un microcontrolador y procesador de señales digitales de la familia de los dsPIC33E (de la empresa *Microchip Technology*). Este *DSP* tiene una arquitectura de código eficiente para lenguajes C y Ensamblador. La programación se realizó en lenguaje C, mediante la herramienta del propio fabricante.

En los puntos siguientes se describen brevemente las principales características de las herramientas de programación y se detalla de manera más extensa el funcionamiento del programa diseñado. El código completo (de los archivos más relevantes) se encuentra integro en el Anexo E.

## 4.1. Herramientas de programación

Durante la etapa de desarrollo del Firmware y su posterior carga en el dsPIC se utilizaron diferentes herramientas.

Para escribir el código del programa (en lenguaje C) se hizo uso del entorno de trabajo MPLAB X IDE (en su versión 5.20), así como del compilador XC16, que utiliza sintaxis y directivas optimizadas para procesadores de 16 bits. También se utilizó el archivo de librerías *"Microchip Libraries for Applications"*, el cual mejora la interoperabilidad cuando se necesita usar más de una librería e incluye proyectos predefinidos y demostraciones de aplicaciones que hacen uso de este tipo de librerías.

Además, se consultaron numerosas guías de desarrollo que ofrece la página web de *Microchip* (https://microchipdeveloper.com) para comprender el funcionamiento de protocolos de comunicación y uso de funciones dentro de ellos.

En cuanto a la herramienta para la carga y depuración del programa en el circuito se utilizó un PICkit 3, también de la empresa fabricante del *DSP*. A pesar de que existen numerosas alternativas para este cometido, se eligió utilizar esta porque ya se disponía de un programador en el laboratorio.

### 4.1.1. MPLAB X IDE

El entorno de desarrollo integrado (IDE) MPLAB X es un software para ordenador que permite desarrollar aplicaciones para la mayoría de microcontroladores y controladores de señal digital (PIC y dsPIC) de la empresa *Microcip Technology*, de manera rápida y sencilla. Ofrece una gran cantidad de funciones para ayudar al usuario a descubrir, configurar, desarrollar y depurar diseños y programas para estos dispositivos de manera que se minimice al máximo el tiempo de desarrollo.

A pesar de que, como la mayoría de programas tan rico en funciones, puede parecer complicado comprender o incluso descubrir todas sus capacidades, este software es muy intuitivo y proporciona numerosas ayudas tanto a nivel visual, como desde las barras de herramientas de la interfaz.

Actualmente (a julio de 2019), la versión más reciente es la 5.20, la cual incluye mejoras significativas con respecto a las anteriores como son un visualizador de datos en tiempo real, una vista de los pines de entrada y salida, enlaces a bibliotecas de software o facilidad de acceso al registro y definición de los bits. El software está disponible para *Windows, Linux* y *Mac*.

El compilador que se utilizó fue el MPLAB XC16, el cual pertenece a la última línea de compiladores en *C*, y que sustituye a los anteriores compiladores HI-TECH y MPLAB C. Está orientado a trabajar con arquitecturas de 16 bits y se integra de manera total con MPLAB para proporcionar una interfaz gráfica completa, de forma que es posible localizar al primer golpe de vista errores de edición, puntos de interrupción o estructuras y tipos de datos entre otras cosas.

Ambas soluciones ofrecen descargas gratuitas con limitaciones: de funcionalidades en el caso del IDE y de niveles de optimización en el caso del compilador.

### 4.1.2. PICkit 3

Para la tarea de la carga del programa y la depuración del mismo, *Microchip* ofrece numerosas soluciones de herramientas para desarrollo que se adaptan a cada aplicación de diseño integrado. Incluyen emuladores en circuito, depuradores en circuito, programadores, etc.

En nuestro caso se utilizó la herramienta PICkit 3 (PG164130). Se trata del depurador/programador hardware de menor coste de *Microchip*, y al igual que otras soluciones como REAL ICE o MPLAB ICD 3, puede programar y depurar microcontroladores PIC y dsPIC de la marca.

El PICkit 3 se conecta a un ordenador mediante la interfaz USB 2.0 y al hardware objetivo mediante la misma interfaz que se encuentra en todos los depuradores/programadores de la empresa, una interfaz de programación en serie en circuito (ICSP), aunque en nuestro caso, la conexión con la PCB se realiza de manera directa, a través de una tira de pines conectados directamente a los puertos PGEC y PGED del dsPIC.


*Figura 4.1. Diagrama de conexiones del depurador/programador PICkit 3*

Esta herramienta no es un programador de producción, por lo que solo debe ser utilizado para fines de desarrollo.

Para obtener más información sobre las herramientas de programación del Firmware, así como consultar guías de uso, se pueden examinar los enlaces a la página de ayuda al desarrollador de *Microchip* (https://microchipdeveloper.com) presentes en la bibliografía del trabajo.

## 4.2. Configuración del dsPIC

Como ya hemos comentado en puntos anteriores, el DSP escogido como núcleo central del hardware es el dsPIC33EP256MU806. Previo a la construcción y el diseño del programa se debe decidir cuál será el modo de funcionamiento del microprocesador, configurando para ello los bits adecuados dentro de los registros de control que conforman las llamadas "palabras de configuración", ya que estas no se podrán modificar en tiempo de ejecución. Estos registros definen parámetros como el modo del oscilador, las protecciones contra lectura/escritura, el estado de algunos periféricos, etc.

Existen para nuestro caso 7 palabras de configuración. En la *Figura 4.2*, se muestra la sección de código que comprende la disposición de los registros de configuración, los cuales se explican a grandes rasgos más adelante.

```
// FGS
#pragma config GWRP = OFF          // General Segment Write-Protect bit
#pragma config GSS = OFF           // General Segment Code-Protect bit
#pragma config GSSK = OFF          // General Segment Key bits

// FOSCSEL
#pragma config FNOSC = FRC         // Initial Oscillator Source Selection bits
#pragma config IESO = OFF          // Two-speed Oscillator Start-up Enable bit

// FOSC
#pragma config POSCMD = HS         // Primary Oscillator Mode Select bits
#pragma config OSCIOFNC = OFF      // OSC2 Pin Function bit
#pragma config IOL1WAY = OFF       // Peripheral pin select configuration
#pragma config FCKSM = CSECMD      // Clock Switching Mode bits

// FWDT
#pragma config WDTPOST = PS32768   // Watchdog Timer Postscaler bits (1:32,768)
#pragma config WDTPRE = PR128      // Watchdog Timer Prescaler bit (1:128)
#pragma config PLLKEN = ON         // PLL Lock Wait Enable bit
#pragma config WINDIS = OFF        // Watchdog Timer Window Enable bit
#pragma config FWDTEN = OFF        // Watchdog Timer Enable bit

// FPOR
#pragma config FPWRT = PWR128      // Power-on Reset Timer Value Select bits
#pragma config BOREN = OFF         // Brown-out Reset(BOR) Detection Enable bit
#pragma config ALTI2C1 = OFF       // Alternate I2C pins for I2C1

// FICD
#pragma config ICS = PGD1          // ICD Communication Channel Select bits
#pragma config RSTPRI = PF         // Reset Target Vector Select bit
#pragma config JTAGEN = OFF        // JTAG Enable bit

// FAS
#pragma config AWRP = OFF          // Auxiliary Segment Write-protect bit
#pragma config APL = OFF           // Auxiliary Segment Code-protect bit
#pragma config APLK = OFF          // Auxiliary Segment Key bits
```

*Figura 4.2. Palabra de configuración para el dsPIC*

- FGS: Define la protección del código fuente. En este caso, se encuentra desprotegido.

- FOSCSEL: Se encarga, entre otras cosas, de configurar la fuente de reloj inicial. La fuente seleccionada para el inicio es el reloj interno.

- FOSC: Registro relacionado con el anterior y que define el modo de selección de la fuente de reloj para el oscilador principal, el cambio de fuente de reloj y la configuración de algunos periféricos. Para nuestro caso, la fuente seleccionada fue un cristal externo en el modo *HS* (de alta velocidad) y con el cambio de fuente habilitado, para poder utilizar posteriormente el PLL asociado a este oscilador.

- FWDT: Asociado al temporizador del perro guardián (*Watchdog Timer*). En nuestro diseño se dejó configurado por defecto y con el temporizador deshabilitado.

- FPOR: Registro para configurar, entre otros, el BOR (*Brown-Out Reset)*, que permite reiniciar el dispositivo si la tensión de alimentación cae por debajo de cierto umbral. Para nuestro caso, fue deshabilitado.

- FICD: Este registro es utilizado para definir cuáles serán los puertos de comunicación con el programador, seleccionando en nuestro caso el par de pines rotulados con el número 1 (*PGC1 y PGD1*).

- FAS: También relacionado con la protección del código y de nuevo en estado desprotegido.

Mientras que para algunos casos se pudo dejar los valores por defecto de los campos, hubo que modificar algunos registros en función de los componentes a utilizar o las conexiones físicas realizadas. Fueron los relacionados con la fuente de reloj y los pines para programación.

## 4.3. Conexión y encendido

Una vez definidos los modos de funcionamiento internos del DSP, pasamos a desarrollar el programa principal que cargaríamos en el procesador, el cual se encargará de capturar las señales y enviarlas por USB al dispositivo *Android*.

El algoritmo mostrado en la *Figura 4.3* describe las funciones que realiza el programa durante la conexión y el encendido del hardware.

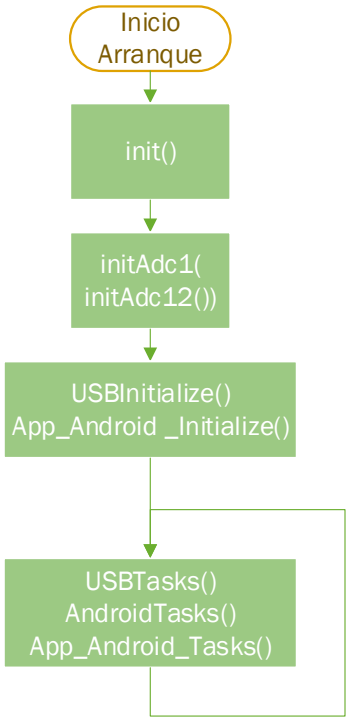



*Figura 4.3. Diagrama de flujo para la conexión y encendido del osciloscopio*

Al conectar la alimentación del circuito comienza la inicialización del sistema, la cual pasa por tres fases.

Durante la primera (llamada a la función '*init'*) se configura el oscilador primario para trabajar a la frecuencia deseada (60 MIPS en nuestro caso) fijando el valor de algunos bits para establecer el PLL principal. También se fija el modo de trabajo del PLL auxiliar para conseguir la frecuencia necesaria para el módulo USB. Se configuran, además, el estado de los puertos (entradas/salidas digitales/analógicas) y de las interrupciones. Debido a la frecuencia del cristal que se utilizó finalmente, no se pudo conseguir valores exactos para las frecuencias de funcionamiento, siendo estas definitivamente: 120,42 MHz para la $f_{osc}$ y 49,15 MHz para el módulo USB. Al comprobar que con estos valores el hardware respondía satisfactoriamente no se vio necesario sustituir el cristal.

La segunda fase (llamadas a las funciones *'initAdc1'* e *'initAdc2'*) se encarga de configurar los conversores analógico-digitales, para trabajar de la siguiente manera:

- Modo de 10 bits de resolución.
- Formato de salida de número entero sin signo (*0000 00dd dddd dddd*).
- Muestreo/conversión automáticos.
- Voltajes de referencia de $AV_{DD}$ y $AV_{SS}$.
- Interrupción cada muestreo/conversión.
- Modo de funcionamiento a 1 Msps.

Como para las escalas de voltaje admitidas no se hace necesaria una resolución superior a los 10 bits, esto permite que el CAD funciones a la máxima velocidad posible que, de nuevo debido a la frecuencia de nuestro cristal, es de 1 millón de muestras por segundo.

Finalmente, durante la tercera fase (llamadas a las funciones *'USBInitialize'* y *'App_Android_Initialize'*) se inicializa el *stack* de USB en el dsPIC para trabajar en el modo de "*host embebido*" y se define la información del accesorio USB utilizado (nuestra PCB personalizada) para su envío al dispositivo *Android*, a la vez que se inicializa la información del driver a utilizar por el dispositivo.

Tras esta etapa inicial, el programa entra en un bucle que se encarga de "mover" el *stack* de la conexión USB a modo de refresco continuo, a la espera de una nueva conexión.

## 4.4. Identificación del estado de la App

Los accesorios que quieren conectarse mediante USB con dispositivos *Android* deben cumplir con el protocolo de accesorio abierto de Android (AOA), el cual define cómo un accesorio detecta y establece la comunicación con un dispositivo con sistema operativo *Android*.

Una vez nuestra PCB detecta la conexión de un dispositivo a través de USB, comienza el proceso de identificación del dispositivo, durante el cual se produce un intercambio de información para determinar si el dispositivo admite el modo de accesorio y, en ese caso, establecer una comunicación con el dispositivo. Durante la conexión inicial, el accesorio debe verificar la identificación del proveedor y del producto del descriptor de dispositivo USB del dispositivo que se ha conectado para realizar estas comprobaciones. Todo este proceso se realiza de manera interna en las funciones de 'USBTasks' y 'AndroidTasks'. A continuación, se ejecuta la función 'App_Android_Tasks', la cual comprueba si ya existe una conexión entre el accesorio (host) y el dispositivo *Android* para seguir con el programa o, en caso contrario, regresa al inicio del bucle con el que concluíamos la explicación anterior.

En el caso de que exista una conexión entre ambos sistemas, el hardware comprueba insistentemente si recibimos la respuesta por parte del dispositivo de que ya está listo para enviar y recibir datos. Cuando recibimos una respuesta, comprobamos de qué información se trata analizando el primer byte recibido. Podemos ver el procedimiento en la *Figura 4.4*.

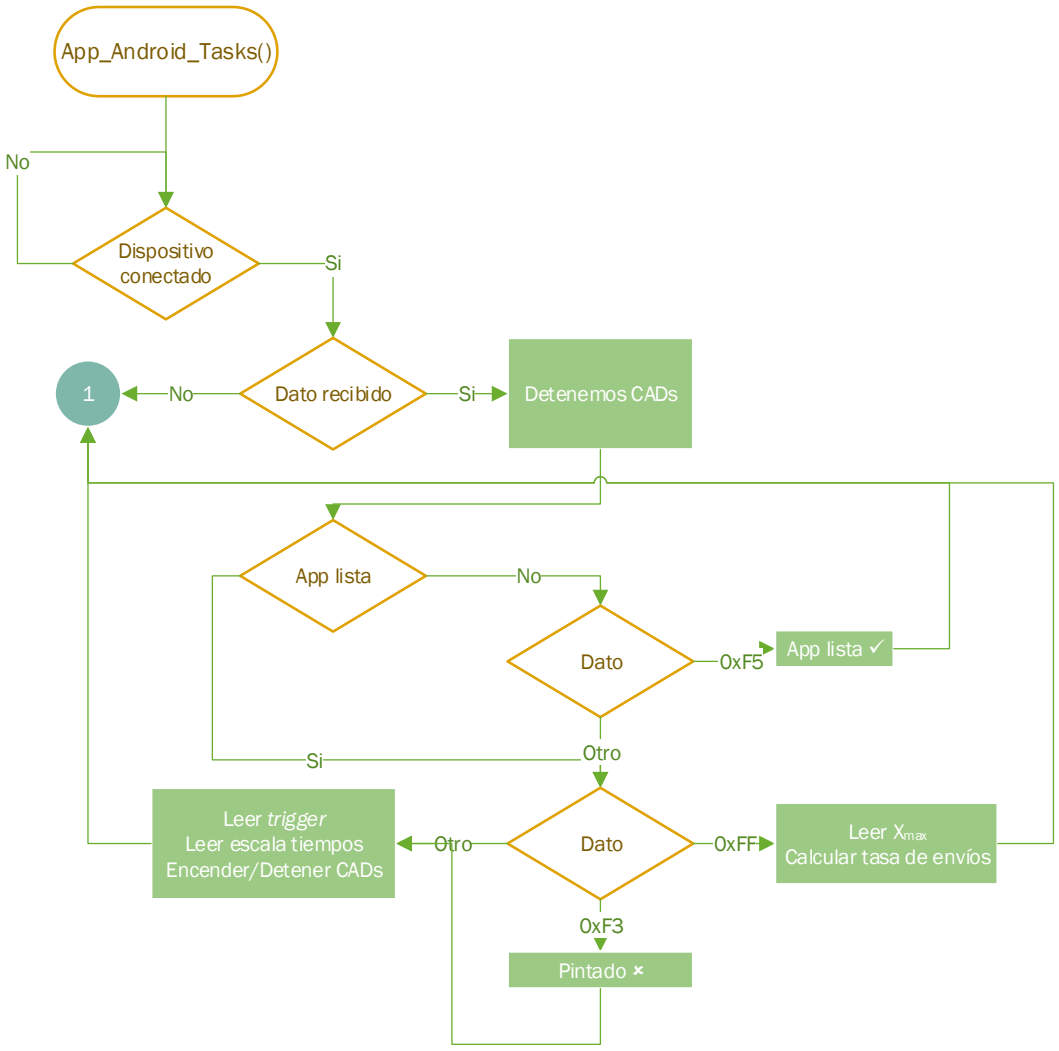



*Figura 4.4. Diagrama de flujo para la lectura de datos*

Si el primer byte recibido contiene el valor 0xF5 (en hexadecimal), eso significa que la App está lista por lo que a continuación, el hardware le solicita el valor del ancho (en píxeles) de la ventana que mostrará la representación de las señales, así como el estado de los controles en la interfaz de la App.

Cuando el primer byte recibido es 0xFF, el hardware detecta que le están enviando el valor del ancho (definido como $X_{max}$), de manera que los siguientes dos bytes leídos corresponderán a las partes alta y baja del dato. Una vez recibidos, compone el dato y realiza una serie de cálculos para definir la tasa de envío de muestras hacia el receptor.

Por otra parte, cuando el primer byte recibido es 0xF3, el dispositivo nos avisa de que ya ha terminado de graficar una pantalla completa y necesita un nuevo envío de muestras.

Finalmente, si el byte que llega no comienza de ninguna de las formas anteriores, el hardware asume que la información que le están enviando se trata del estado de los controles dentro de la aplicación. Estos controles se codifican en un único byte de la siguiente manera.

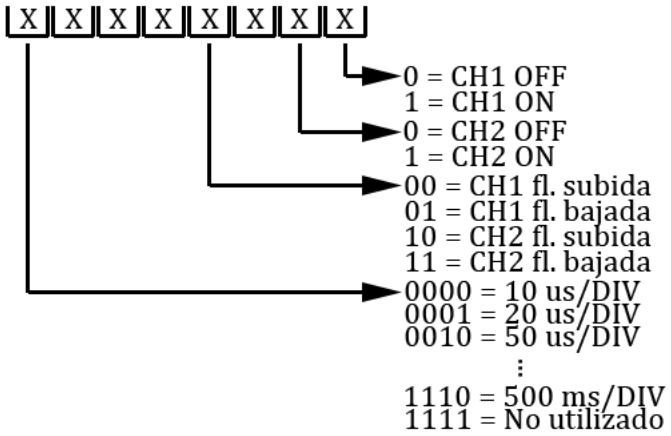



*Figura 4.5. Esquema del byte de estado*

Podemos ver la totalidad de las escalas de tiempo permitidas, así como su correspondencia en bits de selección en la *Tabla 4.1*.

| Bits | Escala | Bits | Escala | Bits | Escala | Bits | Escala |
|---|---|---|---|---|---|---|---|
| **0000** | 10 µs/DIV | **0100** | 200 µs/DIV | **1000** | 5 ms/DIV | **1100** | 100 ms/DIV |
| **0001** | 20 µs/DIV | **0101** | 500 µs/DIV | **1001** | 10 ms/DIV | **1101** | 200 ms/DIV |
| **0010** | 50 µs/DIV | **0110** | 1 ms/DIV | **1010** | 20 ms/DIV | **1110** | 500 ms/DIV |
| **0011** | 100 µs/DIV | **0111** | 2 ms/DIV | **1011** | 50 ms/DIV | **1111** | No utilizado |

*Tabla 4. 1. Bits de selección de escala de tiempo*

Después de recibir y analizar los diferentes campos del byte de estado, el hardware está listo para comenzar la tarea de adquisición de datos para su posterior envío, exceptuando que ninguno de los canales esté aún activo, en cuyo caso volverá a solicitar el valor de estado hasta que esto cambie.

## 4.5. Adquisición de datos

Cuando uno o los dos canales del osciloscopio se encuentran activos, el hardware detiene el proceso de consulta del estado y comienza el de adquisición de los valores de las señales a muestrear. Este procedimiento se realiza dentro de las rutinas de servicio de interrupción de los conversores. Podemos ver una de ellas en la *Figura 4.6* (la otra rutina es análoga y solo cambian las variables de control y asignación, las cuales aparecen destacadas en rojo).

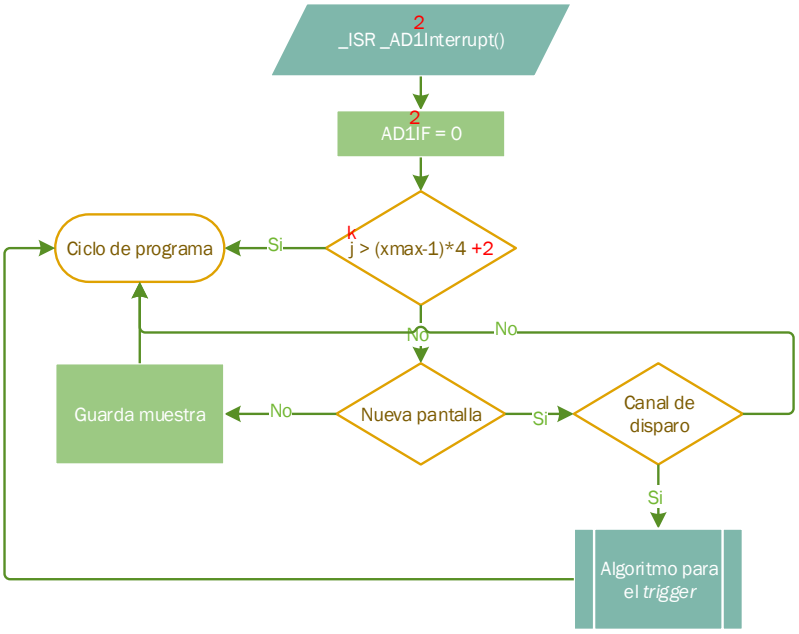



*Figura 4.6. Diagrama de flujo para las rutinas de interrupción de los CAD*

Cada vez que se realiza un ciclo de muestreo/conversión el identificador (*flag*) de notificación de interrupción (AD1IF o AD2IF) del CAD correspondiente se activa, haciendo que el programa detenga su ejecución para atender la rutina de interrupción, y este no regresa a la ejecución hasta que no termina de atenderla.

Lo primero que hacemos es resetear el *flag* de notificación de manera que se pueda detectar una nueva interrupción. A continuación, se comprueba si ya hemos tomado las suficientes muestras como para completar una pantalla en el dispositivo *Android*. En ese caso, regresamos de la rutina sin hacer nada.

En caso contrario, se comprueba si es el primer valor a representar en la pantalla. En caso afirmativo, comprobamos si la fuente de disparo pertenece al canal del CAD que está atendiendo la petición de interrupción, para poder decidir la primera muestra que enviaremos para su representación (*trigger*). Si el canal seleccionado como fuente de disparo es ese mismo, aplicamos el algoritmo del *trigger* (el cual se detalla en la sección posterior) y comenzamos a guardar las muestras. Si el canal seleccionado como fuente de disparo no corresponde con el CAD que lo analiza, este

esperará a que el otro canal dé la señal para comenzar la captura de muestras y hará lo propio.

Una vez la señal es muestreada y convertida a valores digitales (de 0 a 1024), estas muestras se guardan en un buffer de salida, que posteriormente será enviado al dispositivo. Como los datos son de 10 bits, antes de almacenar cada muestra esta se divide en parte alta (MSB) y parte baja (LSB), de modo que se pueda enviar fácilmente mediante las funciones de envío de datos USB, las cuales estén definidas para enviar objetos de tipo byte. La disposición de las muestras en el buffer de salida puede verse en el esquema de la *Figura 4.7*.

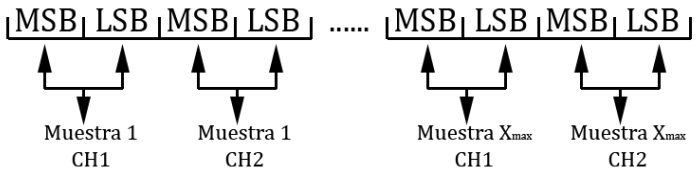



*Figura 4.7. Esquema del contenido del buffer de salida*

Debido a que en escalas de tiempo grandes (en nuestro caso, estas escalas corresponden a los valores mayores a 0,2 ms/DIV) el número de muestras por pantalla supera el número máximo de píxeles disponibles en la ventana de representación marcada por $X_{max}$, antes de enviar esas muestras se produce una operación de diezmado en el hardware, de forma que solo enviaremos el número de muestras suficientes para representar la señal en una pantalla completa cada vez. De esta manera la velocidad de muestreo sigue siendo la máxima posible y el dispositivo *Android* no tiene que descartar muestras repetidas o no válidas.

Para realizar la operación de diezmado se calcula una tasa de envío de datos que corresponderá a la división del tamaño máximo de píxeles de la pantalla del dispositivo entre el número de muestras totales que deberían representarse por pantalla en cada escala de tiempo.

En resumen, el proceso para el envío de tramas hacia el receptor sería el siguiente: se toma un dato, se separa en dos tramas, se almacena en RAM y se toma otro dato. Se itera hasta que el buffer es lo suficientemente grande para completar una pantalla y se envían las muestras.

## 4.6. Algoritmo de disparo (*trigger*)

El algoritmo del *trigger* es el encargado de decidir en qué valor se iniciará la representación en el lado izquierdo de la pantalla, de forma que la señal permanezca estable en la pantalla de acuerdo a la configuración elegida por el usuario desde la App. Para este proyecto, el disparo se realiza de manera automática con un valor aleatorio de la señal, de manera que el usuario solamente puede seleccionar el canal que será la fuente de disparo, así como el flanco de la señal. Podemos ver en la *Figura 4.8*, el diagrama de flujo correspondiente a este algoritmo.

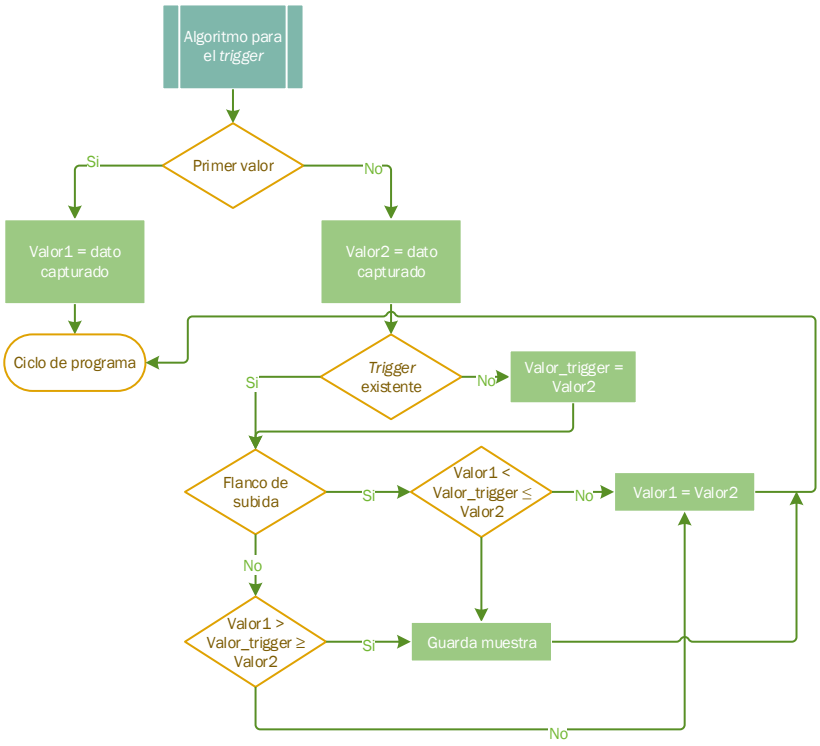



*Figura 4.8. Diagrama de flujo para el algoritmo del trigger*

Una vez se comprueba que el canal que atiende la rutina de interrupción es el escogido como fuente de interrupción, se observa si es la primera muestra que tomamos. En ese caso la guardamos y esperamos a tomar una nueva muestra. Una vez tomamos la siguiente muestra, analizamos si ya tenemos guardado un valor de *trigger*, es decir, si ya hemos representado al menos una pantalla de la señal. En caso contrario, significaría que es la primera pantalla para esa configuración de representación, con lo que guardaríamos ese segundo valor como valor de disparo y se realizaría la comprobación matemática para comprobar el flanco. Esta sería, en caso de flanco positivo de la señal:

$$Valor_1 < Valor_{trigger} \leq Valor_2$$

*Ecuación 4.1. Condición de disparo para flanco de subida de la señal*

Mientras que para el flanco negativo:

$$Valor_1 > Valor_{trigger} \geq Valor_2$$

*Ecuación 4.2. Condición de disparo para flanco de bajada de la señal*

Si la condición no se cumple en cada caso, asignamos el segundo valor al primero y tomaríamos una nueva muestra con el objetivo de volver a comprobar la condición.

En el momento que esta se cumple, guardamos ese valor como primera muestra en el buffer y marcamos la señal de comienzo de pantalla, para que el otro canal (en caso de estar activo) guarde las muestras de manera sincronizada con el disparo del canal activo para el *trigger*.

## 4.7. Envío de los datos

Durante el ciclo principal del programa se pueden enviar datos hacia el dispositivo receptor hasta en tres ocasiones: dos de ellas para realizar una consulta y la otra solamente para enviar datos. Podemos verlo en la *Figura 4.9*.

En cuanto a las primeras, necesitamos realizar una consulta en dos ocasiones:

- Para solicitar el valor de $X_{max}$. En este caso, enviamos dos bytes al dispositivo. El primero contendrá el valor 0xFF, indicando que estamos realizando una consulta. Esto es así debido a que, por el tipo de dato de la conversión del CAD (entero sin signo), los valores de las muestras nunca comenzaran con bits a 1. El segundo byte también contendrá el valor 0xFF, simplemente a modo de indicativo de que la información que solicitamos es el valor máximo del ancho de la ventana.

- Para solicitar el estado de los controles. En este caso también se envían dos bytes hacia el dispositivo *Android*. El primero es de nuevo 0xFF por el mismo motivo descrito anteriormente. El segundo esta vez es 0xF0, como indicativo de que lo que solicitamos es el valor del estado.

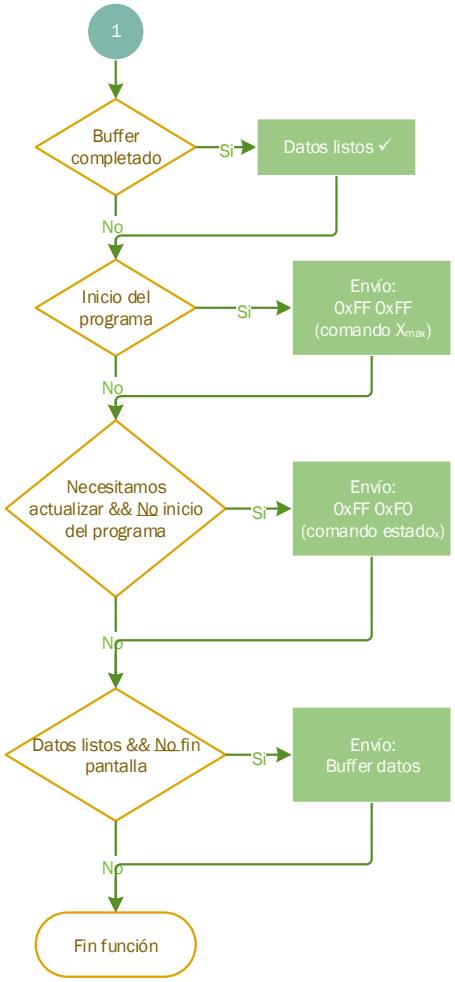



*Figura 4.9. Diagrama de flujo para el envío de datos*

Una vez el buffer de salida está lleno (el número de muestras guardadas es igual o superior al número de píxeles disponibles en la App) se activa la señal que indica que los datos están listos para ser enviados y se procede a su envío, realizando de esta forma la tercera de las maneras de envío de información al receptor. En esta ocasión el número de bytes enviados coincide con el número de muestras almacenadas (que a su vez coincide con el número de píxeles disponibles) multiplicado por cuatro, dos partes por muestra y por canal. El contenido de los bytes es el del buffer de salida y corresponde con los valores convertidos por el CAD.

Inicialmente, una vez concluido el envío de muestras, se consultaba de nuevo el estado para comprobar si había habido algún cambio en los controles antes de enviar de nuevo la siguiente pantalla, pero el tiempo de procesado y posterior graficado de los datos era demasiado alto y no daba tiempo a realizarlo antes de tener que contestar a la solicitud de estado, finalmente se decidió esperar a que el dispositivo nos dijese que ya había terminado de tratar los datos antes de enviarle el siguiente buffer.

Esta parte fue descrita anteriormente en el epígrafe 4.4, con la recepción del byte 0xF3, tras el cual, la aplicación *Android* nos envía el estado de nuevo para así evitar la pregunta y respuesta del mismo.

También se pensó durante la etapa de diseño en enviar muestra por muestra al receptor, pero tras comprender el modelo de transacciones USB y las cabeceras que se añaden a los paquetes, se entendió que se debería enviar el mayor número de datos posible para que la carga útil sea también lo mayor posible.

# Capítulo 5. Programación del Software (App Android)

La tarea de la aplicación *Android* es representar las muestras de las señales capturadas al usuario, quien, además, debe ser capaz de interactuar con la aplicación de manera similar a como lo hiciese con los controles de un osciloscopio convencional.

Esta aplicación recibirá los datos a través del puerto USB del dispositivo, los interpretará y los tratará de forma que se ofrezca una imagen lo más cercana y precisa posible a la señal real de entrada.

En el siguiente capítulo se describirán brevemente las herramientas utilizadas para el desarrollo de la App y se detallará de manera más extensa el funcionamiento del programa.

El código completo de la aplicación se encuentra disponible en el Anexo F.

## 5.1. Herramientas de programación

A la hora de decidir la herramienta que se utilizaría para la programación de la App para *Android*, se tuvieron en cuenta tres alternativas, cada una con un enfoque diferente a la hora de programar.

La primera de ellas fue utilizar el entorno de programación visual del MIT (*Massachusetts Institute of Technology*) "*App Inventor"*. Se trata de un entorno muy intuitivo que permite a todos, incluso niños, crear aplicaciones totalmente funcionales para *Smartphones* y *Tablets*. Permite desarrollar una primera aplicación simple y ponerla en funcionamiento en menos de 30 minutos. Es una herramienta basada en bloques facilita la creación de aplicaciones complejas en mucho menos tiempo que los entornos de programación tradicionales. El principal problema es que no se tiene el control total sobre las partes lógicas y gráficas de la App quedando en cierta parte ocultas en un segundo plano de desarrollo.

Por ese motivo se barajaron los entornos de "*Eclipse IDE*" y "*Android Studio*". Mientras que el primero es puramente un entorno de desarrollo integrado de *Java* al cual se le agregaba un complemento personalizado por *Android* llamado ADT (Herramientas de desarrollo de *Android*) para poder desarrollar Apps, el segundo es el IDE oficial de *Android*, y ofrece herramientas personalizadas para la edición, depuración, pruebas y perfilamiento de códigos para programadores de *Android*.

En este caso las ventajas de *Android Studio* como el *Instant Run*, que permite revisar todos los cambios que hagas en el código casi al instante en emulador, la facilidad y la rapidez de creación de código, la actualización de los cambios gráficos en varias pantallas simultáneamente o el ser un entorno más amigable, hicieron que se optara por este software para el desarrollo de nuestra App.

### *5.1.1. Android Studio*

*Android Studio* es el IDE oficial para el desarrollo de aplicaciones para *Android* y ofrece multitud de funciones que aumentan la productividad del usuario durante la compilación de Apps para *Android*. Entre ellas, incluye un sistema de compilación basado en *Gradle* flexible, emuladores bastante rápidos, *Instant Run*, gran cantidad de plantillas de código, herramientas y *frameworks* de prueba, compatibilidad con C++ y NDK, etc.

Separa el proyecto en carpetas diferentes de forma que queda mucho más organizado dividiendo las partes lógica, gráfica y los *manifests* (archivos que describen información esencial sobre las Apps, necesarios para la herramienta de construcción de la aplicación), y posee un estilo y un formato personalizado que ayuda a identificar partes del código rápidamente.

Durante la realización del proyecto sirvió de gran ayuda el emulador virtual descargado, aunque la mayoría de las pruebas se realizaron en vivo en un dispositivo real conectado por USB o por Wifi al PC.

## 5.2. Requerimientos básicos

Antes de comenzar con la creación del proyecto para la App en la herramienta de diseño, debíamos decidir cuáles serían los requerimientos básicos tanto de la aplicación como de los dispositivos que la soportasen, es decir, para qué tipo de dispositivos programaremos la aplicación y a partir de qué versión de *Android* se daría soporte.

La aplicación sería programada, como se decidió en las especificaciones iniciales del proyecto, para teléfonos y tabletas. En cuanto a la versión de *Android*, se debe tener en cuenta que no todos los usuarios poseen la última versión que se encuentre en el mercado, por lo que debemos asegurar la máxima compatibilidad con el mayor número de dispositivos posibles. Para ello, *Android Studio* nos ofrece una ayuda visual en la que nos informa del tanto por ciento de las personas que tendrán acceso a nuestra App según la versión que utilicemos. Podemos verlo en la *Figura 5.1*.

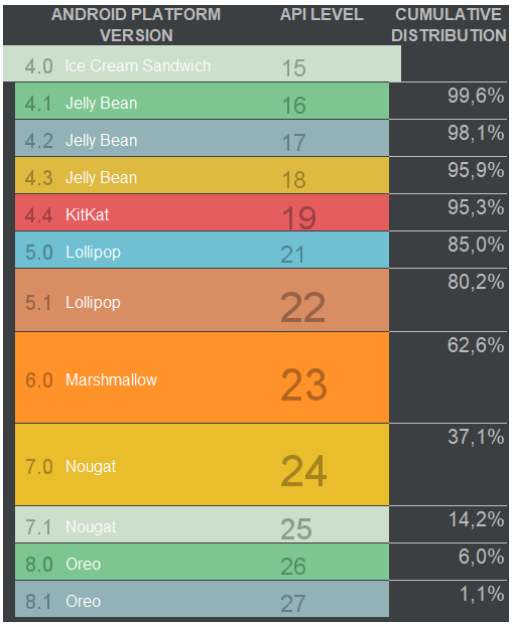



*Figura 5.1. Compatibilidad con usuarios según la versión de Android (Android Studio)*

Teniendo en cuenta esta ayuda y que al protocolo de accesorio abierto necesario para poder trabajar con nuestra PCB a través de USB solo se le da soporte nativo a partir de la versión *Android* 3.1 (API 12), se decidió programar la aplicación para la versión de *Android* 4.0.3 *Ice Cream Sandwich* (API 15), que garantiza que prácticamente el 100% de los usuarios podrán utilizarla.

A pesar de esta última decisión, durante la realización del código se vio necesario utilizar algunos campos que necesitaban de una API mayor para funcionar (API 19: *Android* 4.4 KitKat). Esto implica que solo los dispositivos con versiones iguales o superiores a esta última disfrutaran de la experiencia completa de la aplicación, lo cual se garantiza a un 95,3% de los usuarios.

## *5.3. Activities*

Las *Activities* (actividades, en inglés) son componentes de una aplicación que contienen una pantalla con la que el usuario puede interactuar para realizar acciones como marcar un número de teléfono, hacer una foto o enviar un correo electrónico. A cada actividad se le asigna una ventana en la cual se puede dibujar su interfaz de usuario (UI).

Las aplicaciones, por lo general, contienen numerosas actividades enlazadas entre sí, de forma que una actividad puede a su vez iniciar otra para realizar nuevas acciones. Normalmente, la actividad que se presenta al usuario al iniciar la App por primera vez se especifica como la actividad "principal" (*Main Activity)*, mientras que al resto se le suele dar el nombre de la acción que realicen.

Cada vez que se inicia una actividad nueva, se detiene la anterior, pero el sistema conserva la actividad en una pila, de manera que una vez se inicia la actividad nueva, se la incluye en la pila de actividades y capta el foco del usuario. Esta pila es de tipo LIFO, por lo que, cuando el usuario termina de interactuar con la actividad actual y

presiona el botón 'Atrás', se quita de la pila (y se destruye) y se reanuda la actividad anterior.

Una vez se ha detenido una actividad para iniciar otra, se notifica el cambio de estado a través de métodos "*callback*". Existen diferentes métodos que puede recibir una actividad al cambiar de estado (creación, detención, reanudación o destrucción) y cada uno te ofrece la posibilidad de realizar una tarea específica adecuada para ese cambio de estado. Todos estos cambios de estado conforman el ciclo de vida de una *Activity*, el cual podemos ver en la *Figura 5.2*.

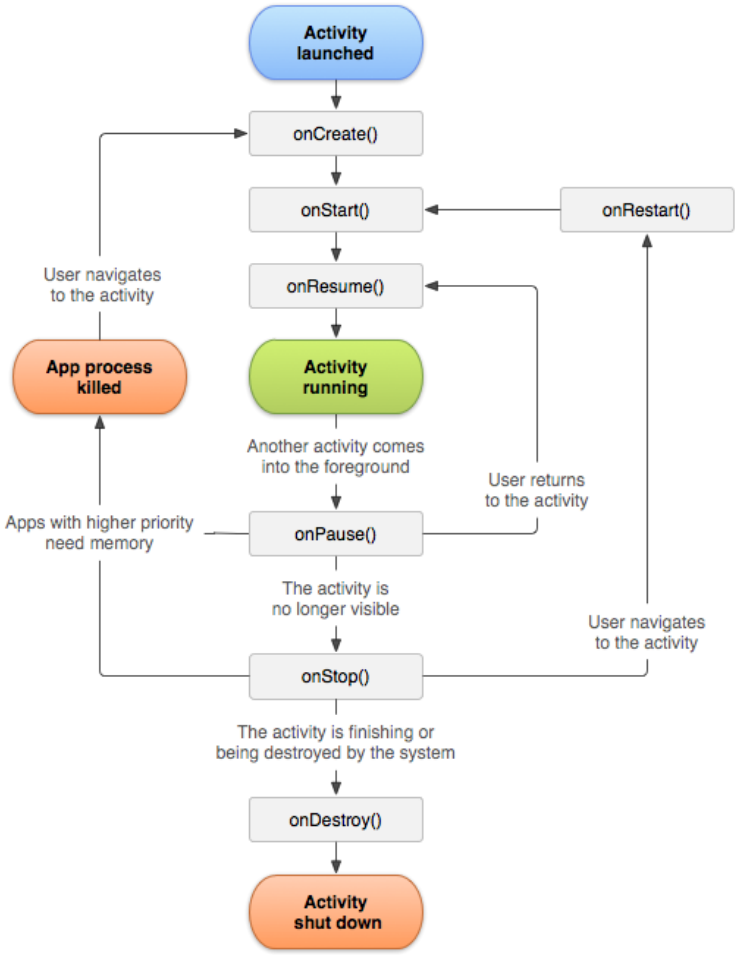



*Figura 5.2. Ciclo de vida de una Activity*

Cada vez que el usuario abre una App lo primero que ocurre es que se ejecuta el método *onCreate*, el cual permite crear la actividad, a continuación, el método *onStart* inicia esa actividad y por último el método *onResume* que permite al usuario visualizar la actividad. En ese punto comienza el funcionamiento del programa hasta que se realice la llamada automática a otro método *callback*.

En el caso de que el usuario minimice la aplicación se ejecutará el método *onPause* haciendo que se pause momentáneamente la actividad pasando de primer a segundo plano. A continuación, se ejecutará el método *onStop*, que deja de hacer visible la aplicación. Con la app detenida, podemos cerrarla completamente (donde

se ejecutará el método *onDestroy*) o bien si se regresa a la aplicación, se ejecutará el método *onRestart*, para que de nuevo se haga visible y operativa al usuario.

En los subapartados siguientes se describe brevemente la interfaz de usuario para cada una de las *Activities* de la App.

### *5.3.1. Main Activity*

Se trata de la actividad principal y se crea cada vez que el usuario ejecuta la aplicación. En ella se muestra el título y logo de la App, así como cierta información relacionada con el presente trabajo. Existen también dos botones que sirven de enlace con las otras dos *Activities* de la aplicación y que están habilitados y accesibles para el usuario desde el inicio.

El botón "Iniciar" lanzará la actividad para control del osciloscopio y la representación de señales, mientras que el botón "Créditos" nos llevará a una actividad puramente informativa.

A continuación, podemos ver una captura de la actividad principal.


*Figura 5.3. Layout de la actividad principal (Main Activity)*

### *5.3.2. Créditos Activity*

Esta actividad únicamente contiene información sobre el TFG, como el título, los autores, las instituciones, etc. Como se trata de bastante información, la actividad es de tipo "*Scroll View*", permitiendo desplazarse a lo largo de la misma para poder visualizar toda la información de una manera más cómoda.

Además, incluye un enlace al documento digital del trabajo para su posible consulta (una vez esté disponible en el repositorio *UVaDoc*) y otros dos enlaces a las páginas web tanto de la escuela como de la universidad a través de las imágenes colocadas en la parte de abajo. Por último, encontramos un único botón en la parte superior

titulado “Volver”, el cual realiza la misma función que el botón “Atrás” de los dispositivos, permitiéndonos regresar a la actividad anterior.

Podemos ver el *layout* de esta actividad en la figura siguiente.


*Figura 5.4. Layout de la actividad de créditos (Creditos Activity)*

### *5.3.3. FullScreen Activity*

Esta es la actividad que muestra las señales en el osciloscopio y permite al usuario controlar la representación de las mismas. Es, probablemente, la actividad donde más tiempo se pase y también la más extensa y compleja. Se encarga de establecer la comunicación con el hardware accesorio a través de USB y gestiona la información (muestras) recibida para su posterior representación.

Pese a que su nombre (*Full Screen*) alude a que será la actividad que se muestre en pantalla completa, como se ha podido observar en las capturas anteriores el resto de actividades también se muestran en pantalla completa. Este cambio se decidió después de haber creado el proyecto en *Android Studio* y se conservó el nombre hasta el final.

La interfaz de usuario, cuyo *layout* podemos ver en la *Figura 5.*5, cuenta con un gran espacio para la zona de representación de las señales, la cual se adaptará al máximo espacio disponible en pantalla en cada tipo de dispositivo y para cada resolución de pantalla. A su derecha, podemos encontrar los diferentes controles para seleccionar las escalas de voltaje para cada canal, la escala de tiempos (para ambos canales simultáneamente) y la selección del canal y flanco de disparo de las señales. Por otra parte, bajo la ventana de representación, encontramos cuatro indicadores que reflejan los voltajes y frecuencias de las señales en pantalla, así como los botones para activar o desactivar los canales, los cuales, por defecto, se encuentran desactivados. Finalmente, en la parte inferior derecha de la pantalla encontramos el botón titulado “Funciones”, el cual despliega un pequeño menú flotante que nos

permite seleccionar entre una variedad de funciones como realizar una captura de pantalla de las señales para guardarla en la galería, o guardar el valor de las señales en un archivo de texto para su posterior análisis y visualización mediante otras herramientas o programas informáticos. En este menú podrían añadirse nuevas funcionalidades en versiones futuras.

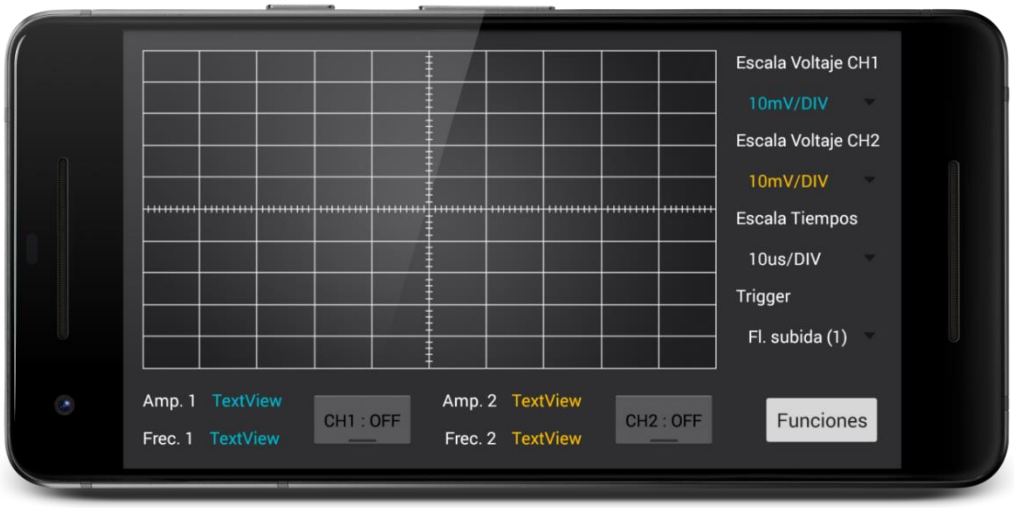



*Figura 5.5. Layout de la actividad para el osciloscopio (FullScreen Activity)*

## 5.4. Conexión y encendido

Como hemos visto anteriormente, cada vez que la aplicación se ejecuta se crea la actividad principal, desde la cual el usuario accede al control del osciloscopio pulsando el botón "Iniciar". Debido a la sencillez de las otras dos actividades (*Main Activity* y *Créditos Activity*) nos centraremos en el ciclo de vida de la *FullScreen Activity*, por ser, además, la que desarrolla el programa principal.

Cuando el usuario conecta el accesorio USB a un dispositivo *Android*, el sistema comprueba si la aplicación reconoce el accesorio conectado para, en tal caso, configurar la comunicación con dicho accesorio. Esto se realiza mediante un filtro de intención colocado en el archivo *"AndroidManifest"*, de forma que la aplicación detecta automáticamente el accesorio, solicita al usuario permiso para comunicarse con él y configura la comunicación. De esta manera, tan solo con conectar nuestro dispositivo con la PCB fabricada, se nos pedirá que iniciemos la aplicación en cuestión.


*Figura 5.6. Mensaje emergente al conectar la PCB al dispositivo Android*

Podemos ver el diagrama de estados que define la conexión y encendido del sistema en la aplicación en la *Figura 5.7*.

Una vez el usuario lanza la aplicación y pulsa en "Iniciar" (mostrado en la *Figura 5.3*), el método *onCreate* de la actividad del osciloscopio se ejecuta, creando dicha actividad y configurando la comunicación con el hardware. Para ello, crea un elemento que registra las conexiones y desconexiones del USB abriendo o cerrando la comunicación con el accesorio correspondiente. Una vez abre esa comunicación, se crea un nuevo subproceso de trabajo (círculo 1 en la *Figura 5.7*) para leer los datos enviados por el hardware y tratarlos. Este subproceso será explicado en secciones posteriores.

A continuación, carga el *layout* de la actividad para mostrar la interfaz al usuario, y crea las relaciones entre la parte lógica (el código) y los objetos de la parte gráfica (los controles de la interfaz) de forma que cada cambio realizado por el usuario tenga asociado un evento en la programación. Por último, crea los métodos de "escucha" (*listeners*) para los objetos mencionados anteriormente, calcula el valor de $X_{max}$ necesario para nuestro hardware y notifica a este último de que ya está listo enviando el comando correspondiente.

Tras el método *onCreate*, como ya hemos visto en el ciclo de vida de una actividad, se ejecuta el método *onStart*, que inicia la actividad. En nuestro caso este método pasa desapercibido, ya que no realizamos ninguna otra tarea en él. Y ya después, se ejecuta el método *onResume*, el cual comprueba si ya estamos conectados con el accesorio y, en caso de no estarlo (porque vengamos de tener la aplicación detenida en segundo plano), abre la comunicación con él. Tras ese punto comienza la ejecución del programa, el cual se va detallando en los siguientes epígrafes.

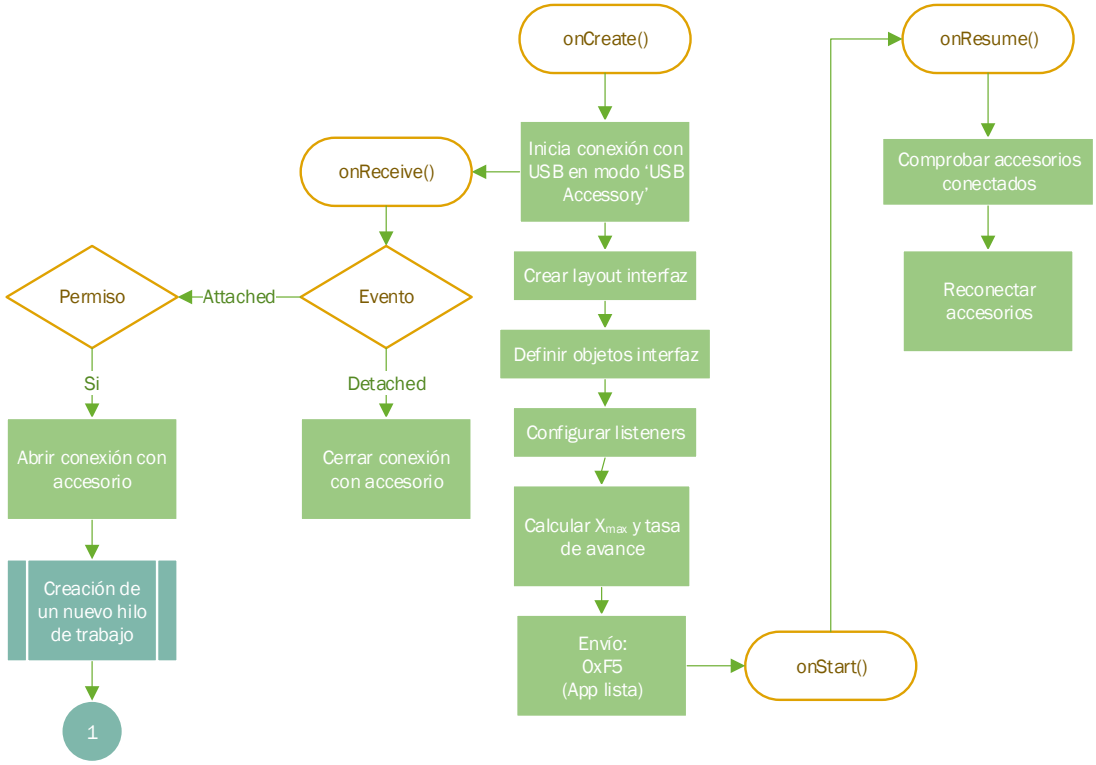



*Figura 5.7. Diagrama de flujo durante el inicio y la conexión de la App*

## 5.5. Listeners de los objetos

Los denominados *listeners* de los elementos de control de la interfaz son, en realidad, comprobaciones dentro del código que se ejecutan cada vez que el usuario interacciona con estos elementos para poder comprobar, por ejemplo, en qué posición se encuentra un selector o si se ha pulsado un botón.

En nuestro caso, los eventos que debemos comprobar son:

- Activación/desactivación de los canales: Se realiza mediante la función *setOnClickListener*, la cual comprueba si se ha pulsado o no el botón y fija su estado en ON/OFF. Cada vez que se pulsa un botón (se quiere activar un canal), se comprueba que la fuente de disparo corresponda a ese canal o, en caso contrario, que, si corresponde al otro canal, este esté ya activo. Si se cumplen estas condiciones, activamos el canal y registramos el evento en el registro (byte) de estado, para notificárselo al hardware cuando lo necesite.

- Selección de escalas (voltajes, tiempos y *trigger*): Se realiza mediante la función *setOnItemSelectedListener*, la cual comprueba la posición en la que se encuentra el selector cada vez que el usuario pulsa en él. En nuestro caso, guardamos la posición del selector para futuros cálculos y registramos el evento en el byte de estado.

## 5.6. Envío/recepción de datos

Una vez la aplicación está lista para enviar y recibir datos, comienza el flujo principal del programa, donde la aplicación, gracias al subproceso de trabajo creado durante la conexión con el accesorio, comprueba insistentemente si existe alguna transmisión entrante que leer. Si consigue leer algún dato desde el hardware, lo analiza y responde enviando la información oportuna en caso de necesitarlo.

A continuación, se explica de manera más detallada el uso de las funciones de envío de datos y la rutina de trabajo de recepción de datos. Debemos notar que la información (comandos y solicitudes) intercambiada es la misma que ya hemos analizado en el capítulo pasado, solo que desde el lado del dispositivo *Android*. Debido a que la función para el envío es más sencilla se comenzará con ella.

### 5.6.1. Envío de datos

Esta función se utiliza cada vez que la aplicación necesita responder a alguna petición de información por parte del hardware accesorio, o bien, para notificar de algún cambio importante en el entorno de la aplicación. En concreto se hace uso de esta función hasta en cuatro ocasiones:

- Para notificar la disponibilidad. Una vez la aplicación ha calculado los datos necesarios para poder realizar las operaciones sin problema, esta lo notifica al accesorio enviando el comando 0xF5.

- Para enviar el valor de $X_{max}$. Cuando la aplicación recibe la solicitud para el envío del valor máximo en píxeles de la ventana de visualización, esta separa el dato en dos bytes (debido a que este valor casi siempre es superior a $2^8$ píxeles) y los envía hacia el host.

- Para enviar el estado de los controles. Cada vez que se recibe la solicitud de información del estado de botones y selectores, y cada vez que se ha terminado de graficar una pantalla, la aplicación envía el byte de estado.

- Para informar del fin del trazado. Tras haber terminado de representar el trazado de una o las dos señales, la aplicación envía hacia la PCB dos bytes: el primero con el valor 0xF3 (indicando este suceso), y el segundo con el estado de los controles, para aprovechar el envío y evitar una nueva transferencia.

### 5.6.2. Recepción de datos

La recepción de los datos, como comentábamos anteriormente, se realiza dentro de una rutina o subproceso de trabajo distinto al principal, como se muestra en la *Figura 5.8*. Este hilo se crea y ejecuta cuando la aplicación y el hardware establecen la conexión USB, y permanece activo todo el tiempo hasta el cierre de la conexión, comprobando continuamente si existen datos que leer en el flujo de datos de entrada y, en caso de existir, tomando las decisiones oportunas.

Una vez se logra leer algún dato lo primero que se hace es examinar el primer byte de la transmisión, para ver si este es un comando (empezará por 0xFF) o es un dato muestreado (no comenzará por 0xFF).

En el caso de que lo que se recibe sea un comando, se analiza el valor del segundo byte para comprobar de qué comando se trata. Puede haber dos posibilidades:

- El segundo byte es 0xFF. En esta ocasión nos están solicitando el valor de $X_{max}$.

- El segundo byte es 0xF0. Para este caso lo que nos solicitan es el estado de los controles.

En el caso de que el primer byte tenga un valor distinto, significará que lo que nos están enviando son los valores de las muestras de las señales para su posterior representación en el dispositivo. En este caso, las comprobaciones que se hacen son diferentes y ya tienen que ver con el análisis y el tratamiento de esas muestras.

Lo siguiente que se hace es comprobar si ya se ha calculado el alto (en píxeles) de la ventana de representación, para poder adecuar los valores de las muestras a cada pantalla y representarlas de manera efectiva. Recordemos que ya habíamos calculado el largo (ancho) de esa ventana en pasos anteriores, sin embargo, para el cálculo del alto se debía esperar un poco más, ya que a menudo hasta el final de la ejecución del método *onResume* este dato no estaría disponible, o sería erróneo.

Después de obtener este dato, en el caso de que no lo tuviésemos ya, se realiza el cálculo de las constantes de escalado para las muestras y se asignan a las variables correspondientes según la posición en la que se encuentre el selector de escala de voltajes. A continuación, se comprueba si la que estamos analizando es la primera muestra para una nueva pantalla. En ese caso se guarda como posición inicial en el trazo que luego representaremos. Además, asignamos los valores para iniciar el cálculo de la frecuencia de las señales.

Si no es la primera muestra para una nueva pantalla, simplemente registramos el valor en el trazo correspondiente a cada canal y seguimos calculando amplitudes y frecuencias, comprobando si el valor recibido es mayor o menor que el anterior, y si hemos vuelto a pasar por la misma pareja de valores respectivamente para los cálculos anteriores.

En el caso de que ya hayamos leído todo el buffer de muestras que nos enviaron, o que ya tengamos suficientes muestras para representar una pantalla, se crea un nuevo subproceso (círculo 2 en el diagrama de la *Figura 5.8*), en este caso para dibujar el/los trazo/s obtenido/s en la ventana de visualización. Esto es necesario debido a que el modelo de subproceso único que utiliza *Android* para el hilo de UI (Interfaz de Usuario) dice que no se puede acceder al paquete de herramientas de la UI de *Android* desde fuera del subproceso de UI. De otra manera, la capacidad de respuesta de la UI se vería afectada y la aplicación se detendría.

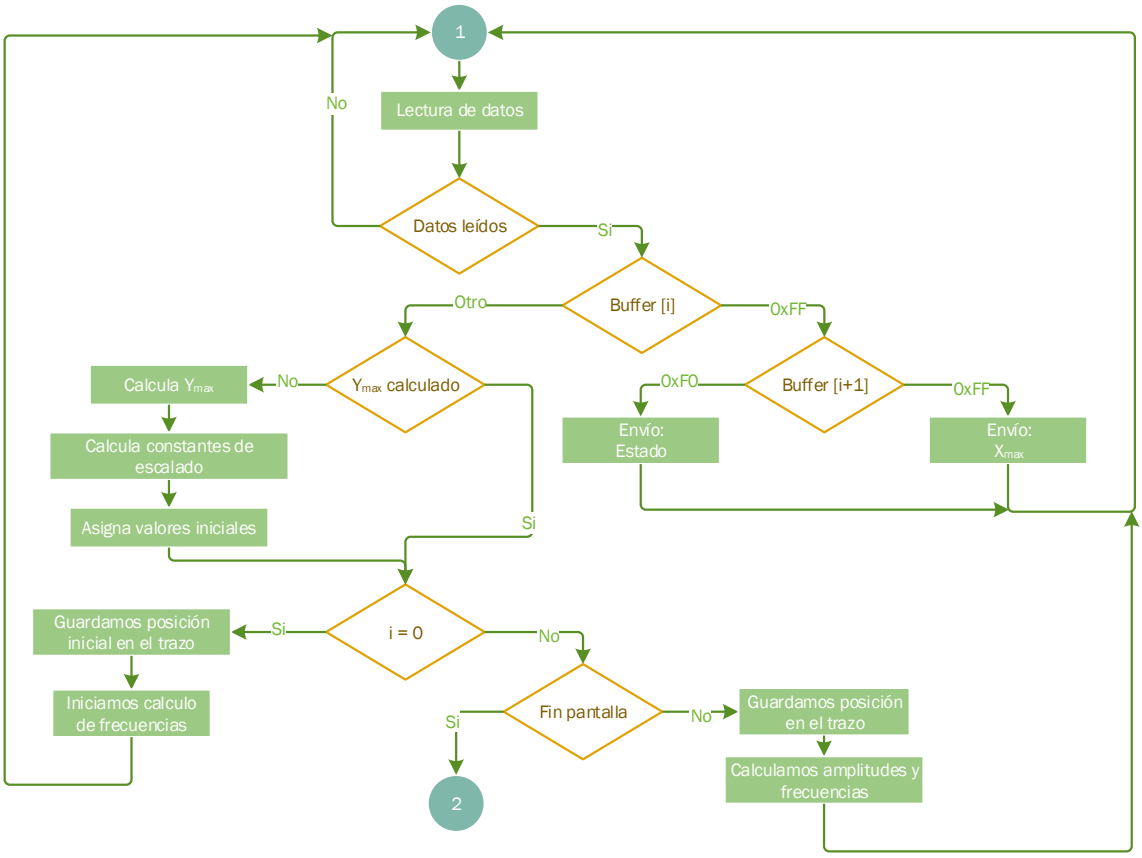



*Figura 5.8. Diagrama de flujo para el subproceso de recepción de datos*

## 5.7. Trazado de las señales

Como indicábamos antes, una vez tenemos suficientes muestras para representar una pantalla, se inicia el proceso de trazado de las señales, como se muestra en la *Figura 5.9*.

Lo primero que se hace es llamar a una función para limpiar la pantalla de modo que no sobrescribamos varios trazos en ella, liberando así memoria en el dispositivo y consiguiendo una visualización mucho más clara para el usuario.

A continuación, se comprueba qué canales están activos, para representar solo una señal en pantalla, o las dos señales simultáneamente. Después se finaliza el cálculo de amplitudes, comprobando los valores máximos y mínimos capturados, y de frecuencias, comprobando cuándo (en el caso de que haya sucedido) se ha vuelto a pasar por el mismo par de valores de la señal. Una vez obtenidos estos valores, se muestran por pantalla utilizando los visores para texto (*TextView*) debajo de la ventana de visualización.

Finalizado el trazado de las señales, se envía una notificación al hardware indicando la finalización de una pantalla.

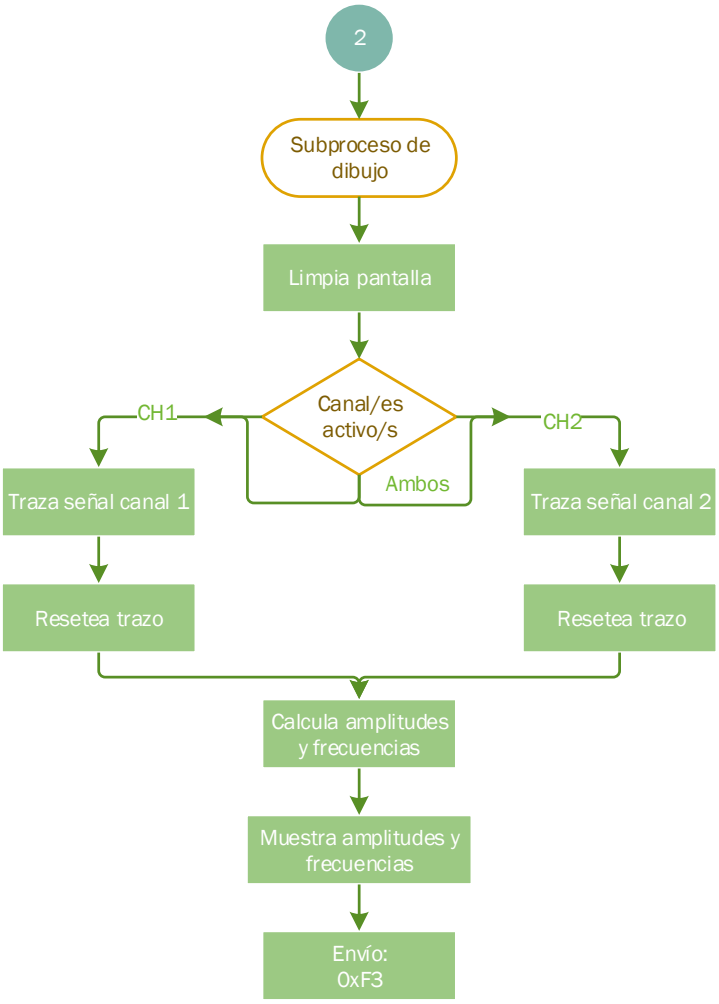



*Figura 5.9. Diagrama de flujo para el subproceso de trazado de las señales*

Como ya hemos visto en la sección dedicada al Hardware, los datos convertidos por los CADs del dsPIC están comprendidos entre 0 y 1024, correspondiendo a los valores de 0 V y 3,3 V respectivamente. El problema es que en la pantalla del dispositivo no tendremos una altura exacta de 1024 píxeles en la ventana de visualización, ya que cada dispositivo es diferente, por lo que tenemos que adaptar las muestras para obtener la representación correcta.

Para esto, debemos conocer el alto de nuestra pantalla y la posición del selector de escalas de voltaje, y así conocer qué constante utilizar para la adaptación de las muestras. La ecuación que define el valor en pantalla de los datos recibidos es la siguiente.

$$Valor_{Pantalla} = Ke_{1,2,3} \cdot (Muestra - 512) + \frac{Y_{max}}{2}$$

*Ecuación 5.1. Fórmula para el cálculo de los valores en pantalla*

Donde la Ke es la constante de escalado según la escala de voltajes escogida.

Como la selección del factor de atenuación/amplificación no se realiza de manera automática, sino que debe seleccionarse físicamente mediante el interruptor habilitado para tal efecto en la PCB del hardware, debemos asegurarnos que el valor del selector de escala de voltaje para los canales corresponde con la posición correcta del interruptor del circuito. En la *Tabla 5.1* podemos ver estas relaciones para poder asegurar que el proceso de trazado será correcto.

| Posición del interruptor | Escala de voltajes | Rango de valores representados |
|---|---|---|
| 1 | 10 mV/DIV | (-50 mV a 50 mV) |
| | 20 mV/DIV | (-100 mV a 100 mV) |
| | 50 mV/DIV | (-250 mV a 250 mV) |
| 2 | 0,1 V/DIV | (-500 mV a 500 mV) |
| | 0,2 V/DIV | (-1 V a 1 V) |
| | 0,5 V/DIV | (-2,5 V a 2,5 V) |
| 3 | 1 V/DIV | (-5 V a 5 V) |
| | 2 V/DIV | (-10 V a 10 V) |
| | 5 V/DIV | (-25 V a 25 V) |
| 4 | 10 V/DIV | (-50 V a 50 V) |
| | 20 V/DIV | (-100 V a 100 V) |
| | 50 V/DIV | (-250 V a 250 V) |

*Tabla 5.1. Relación de valores de la escala de voltaje con el interruptor de la PCB*

En caso de que se seleccione una escala de voltaje sin haber colocado el interruptor en la posición correspondiente, la medida que nos ofrecería la representación, tanto visual como numérica sería engañosa.

## 5.8. Funciones

El botón "Funciones" se encuentra alojado en la esquina inferior derecha de la pantalla y permite al usuario seleccionar una o varias funciones de entre las que se muestran en un menú flotante que se despliega al pulsar el propio botón.

Podemos ver este menú en la *Figura 5.10*.


*Figura 5.10. Menú flotante para funciones en la App*

La idea es que el osciloscopio realice las funciones básicas para las especificaciones definidas y se vayan incluyendo funciones a medida que se vayan desarrollando.

La versión 1.0 del proyecto incluye por el momento dos funciones:

- Guardar captura de pantalla. Permite al usuario guardar una captura de la ventana de visualización de las señales en la galería. Cuando el usuario selecciona esta opción, se crea un objeto de tipo *Bitmap* a partir de la ventana de visualización y posteriormente este objeto es guardado en un archivo con extensión ".png" y titulado según la fecha y hora en la que se guardó.

- Guardar archivo de datos. Si el usuario selecciona esta opción, se crea un archivo de texto con extensión ".txt" que incluye los valores de *X* e *Y* para cada muestra de cada canal, de forma que pueda utilizarse posteriormente para su análisis. El nombre que se le da al archivo es, igual que en el caso anterior, la fecha y hora en que se creó, y el archivo se guarda en el directorio raíz de la tarjeta SD (en el caso de que se disponga de una), notificando al usuario mediante un mensaje la ruta donde se guardó en su dispositivo. El formato de este archivo guarda en sus primeras posiciones los valores de la escala de voltajes para el canal 1, la escala de voltajes para el canal 2 y la escala de tiempos utilizada (todo ello referenciado a la posición en el selector de la aplicación). Seguido continúan las muestras en orden según: valores *X* e *Y* de la muestra 1 del canal 1, valores *X* e *Y* de la muestra 1 del canal 2, y así sucesivamente. Podemos ver un ejemplo de este fichero en la *Figura 5.11*.

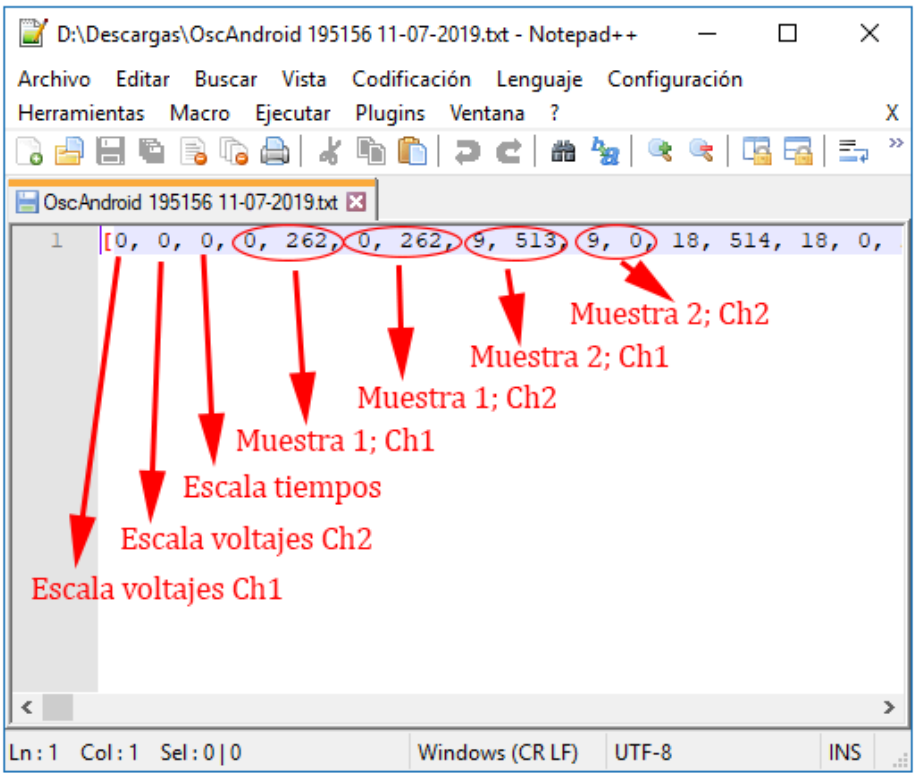



*Figura 5.11. Fichero de prueba generado por la aplicación explicado*

En la *Tabla 5.2* se observa la correspondencia de los tres primeros valores del archivo con las escalas determinadas en cada caso.

| Valor | 1ra y 2da posiciones en el archivo | 3ra posición en el archivo |
|---|---|---|
| **0** | 10 mV/DIV | 10 µs/DIV |
| **1** | 20 mV/DIV | 20 µs/DIV |
| **2** | 50 mV/DIV | 50 µs/DIV |
| **3** | 0,1 V/DIV | 100 µs/DIV |
| **4** | 0,2 V/DIV | 200 µs/DIV |
| **5** | 0,5 V/DIV | 500 µs/DIV |
| **6** | 1 V/DIV | 1 ms/DIV |
| **7** | 2 V/DIV | 2 ms/DIV |
| **8** | 5 V/DIV | 5 ms/DIV |
| **9** | 10 V/DIV | 10 ms/DIV |
| **10** | 20 V/DIV | 20 ms/DIV |
| **11** | 50 V/DIV | 50 ms/DIV |
| **12** | - | 100 ms/DIV |
| **13** | - | 200 ms/DIV |
| **14** | - | 500 ms/DIV |

*Tabla 5. 2. Correspondencias con escalas de voltajes y tiempos para las primeras posiciones en el archivo*

# Capítulo 6. Pruebas y análisis

En el presente capítulo se describirán las pruebas que se diseñaron para probar el funcionamiento del osciloscopio una vez que tanto el hardware como la aplicación estuvieron totalmente operativos, ya que inicialmente surgieron diferentes problemas que se fueron solucionando depurando paso a paso el código o estudiando pormenorizadamente cada sección del mismo.

El equipo utilizado para realizar las diferentes pruebas se cita a continuación.

- Osciloscopio digital *Tektronix TDS 320*, con un ancho de banda de 100 MHz y una tasa de muestreo de hasta 500 Msps
- Generador de funciones *TTi TG210*, con un rango de frecuencias desde 0.02 Hz hasta 2 MHz y un rango de amplitud desde los 60 mV hasta los 20 V
- PCB prototipo de este proyecto
- Fuente de alimentación de 5 V y 2 A
- Smartphone Samsung Galaxy S3 con *Android 4.4.4*
- Sondas de medida con atenuaciones de 1x y 10x

## 6.1. Pruebas

Para analizar el sistema se colocó a la entrada de cada canal del circuito una señal de amplitud y frecuencia conocida, a la cual se le irían variando diferentes aspectos como la forma de onda de la señal, la amplitud o la frecuencia. A la vez que introducimos la señal en el circuito, esta es visualizada con un osciloscopio comercial para comprobar las diferencias que existen entre ambos sistemas.

En la *Figura 6.1* podemos ver, a modo de ejemplo, el esquema del montaje para las medidas realizadas durante este capítulo.

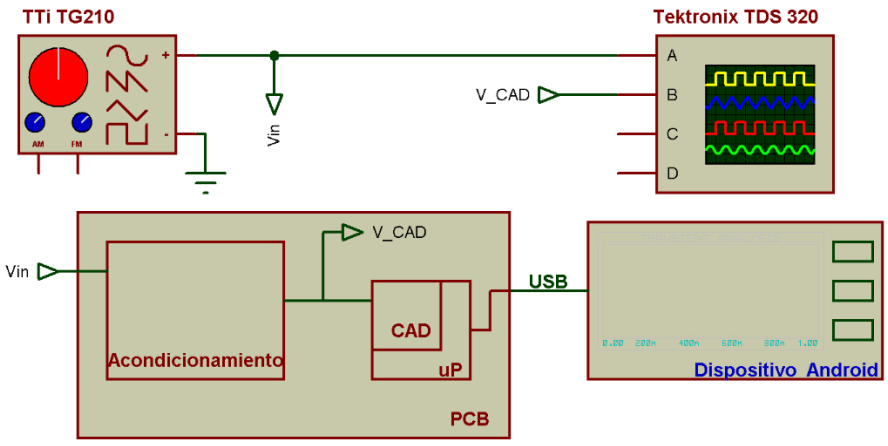



*Figura 6.1. Esquema del montaje para las pruebas*

Para el análisis de resultados debemos tener en cuenta que la sonda que se utilizó para introducir la señal en el circuito era x1, mientras que las sondas utilizadas para la representación en el osciloscopio eran ambas x10. Por este motivo la amplitud que vemos en la aplicación *Android* se ve atenuada en un factor $\frac{1}{10}$.

**Prueba 1: Señal sinusoidal [1 V, aprox. 20 KHz]**

Para la primera prueba se decidió introducir una señal sinusoidal de amplitud 1 V y una frecuencia de unos 20KHz. Ambos valores están en un rango adecuado para realizar prácticas docentes en laboratorio.

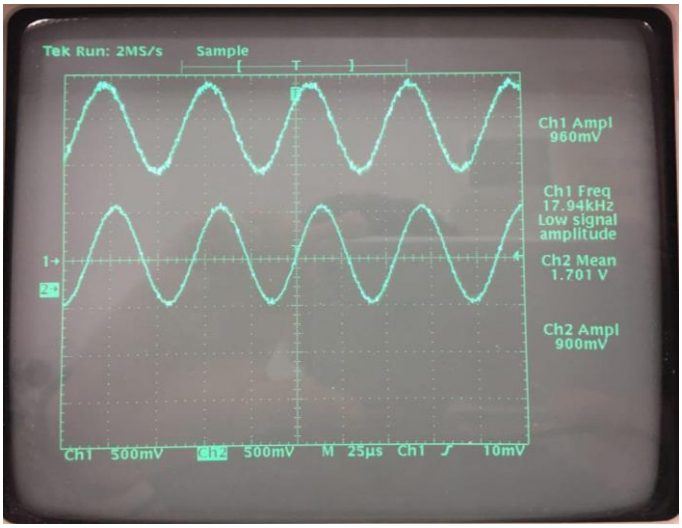



*Figura 6.2. Señal de entrada de prueba (abajo) y señal a la entrada del CAD 1 (arriba) en el osciloscopio comercial*

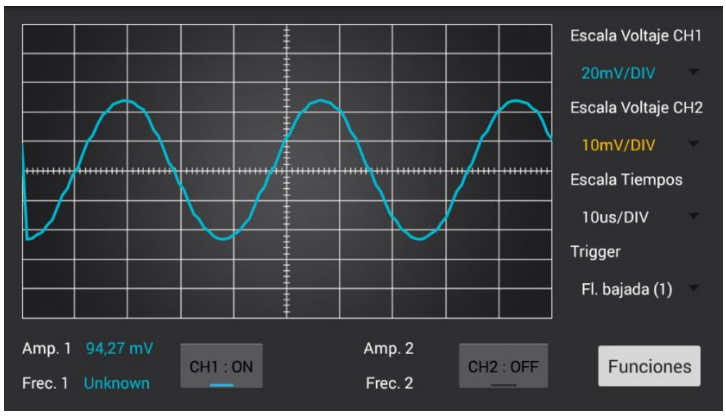



*Figura 6.3. Señal de entrada de prueba de 1 V (escalada) en el osciloscopio del proyecto*

| Equipo | *Tektronix TDS320* | Prototipo desarrollado |
|---|---|---|
| **Amp. Ch1** | 960 mV | 94,27 mV |
| **Frecuencia** | 17,94 KHz | ~27,03 KHz |

*Tabla 6.1. Resultados en ambos equipos de medida para la prueba 1*

Como podemos observar en los datos de la *Tabla 6.1*, la amplitud de la señal es muy parecida a la de la señal de entrada, sin olvidar que se encuentra escalada por el

factor de $\frac{1}{10}$ debido a la sonda de entrada. Esto sería adecuado, salvo porque no se está efectuando la atenuación por el factor ½ que aplicaba la etapa de entrada del circuito mediante la serie de resistencias de 510 KΩ (*Figura 3.2*).

En el caso de la frecuencia vemos dos problemas: el primero es que la aplicación no logra calcular un valor concreto, ya que no coinciden los valores de comparación (el algoritmo que calcula la frecuencia en el programa de la App compara pares de valores iguales, y al no pasar la señal por los mismos puntos exactamente, no es capaz de mostrar el resultado), y el segundo es que el valor que ofrece la visualización dista bastante del valor real de frecuencia de la señal. Podemos ver este efecto midiendo gracias a la rejilla de la aplicación.

Todos estos resultados se explicarán detalladamente en el epígrafe siguiente.

**Prueba 2: Señal sinusoidal [1 V, aprox. 5 KHz]**

Para la segunda prueba se mantuvo la forma de onda y la amplitud y solo se varió la frecuencia de la misma, reduciendo esta en una década y aproximándola a los 5 KHz.

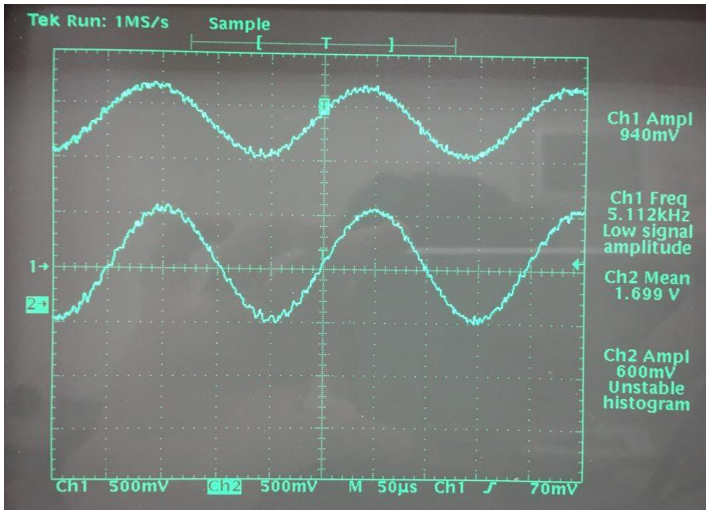



*Figura 6.4. Señal de entrada de prueba 2 (abajo) y señal a la entrada del CAD 1 (arriba) en el osciloscopio comercial*

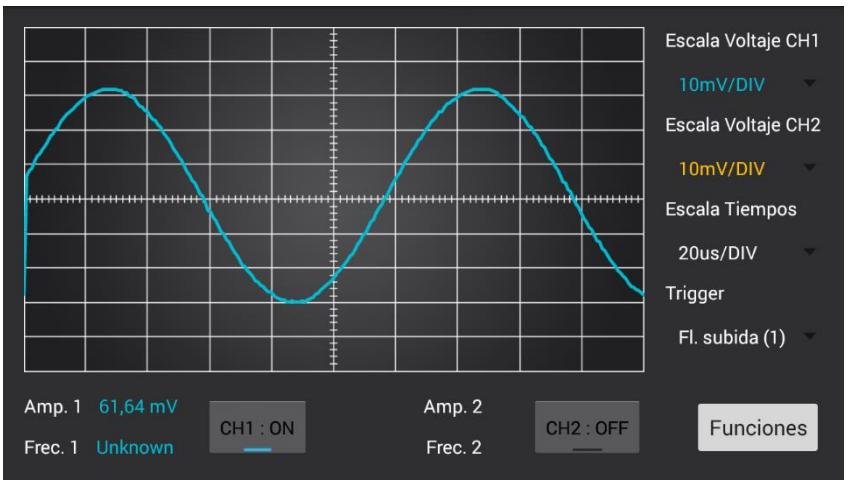



*Figura 6.5. Señal de entrada de prueba 2 de 1 V (escalada) en el osciloscopio del proyecto*

| Equipo | *Tektronix TDS320* | Prototipo desarrollado |
|---|---|---|
| **Amp. Ch1** | 1 V | 61,64 mV |
| **Frecuencia** | 5,11 KHz | ~8,33 KHz |

*Tabla 6.2. Resultados en ambos equipos de medida para la prueba 2*

En este caso vemos que, para la amplitud de la señal en la aplicación, aparece el mismo efecto de atenuación de la sonda (10 veces menor que la señal en el osciloscopio), pero, además, sí que aparece el efecto de atenuación por ½ que mencionábamos en la prueba anterior.

En cuanto a la frecuencia, de nuevo el valor es desconocido, aunque realizando un cálculo aproximado a través de la visualización, vemos que también dista bastante de la frecuencia de la señal de entrada.

**Prueba 3: Señal sinusoidal [10 V, aprox. 20 KHz]**

Para esta prueba lo que se varió fue el valor de amplitud de la señal de entrada, aumentándola en un factor x10. La frecuencia se fijó de nuevo en unos 20 KHz. Por último, en la PCB se seleccionó la posición de escala adecuada en el interruptor.

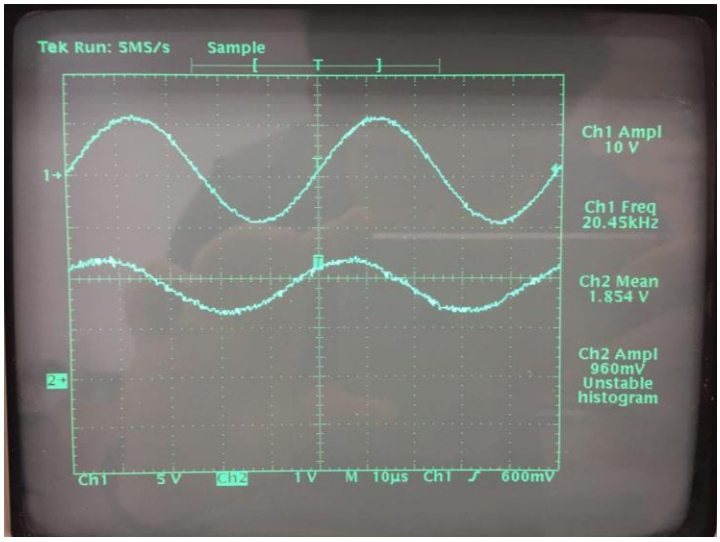



*Figura 6.6. Señal de entrada de prueba 3 (abajo) y señal a la entrada del CAD 1 (arriba) en el osciloscopio comercial*

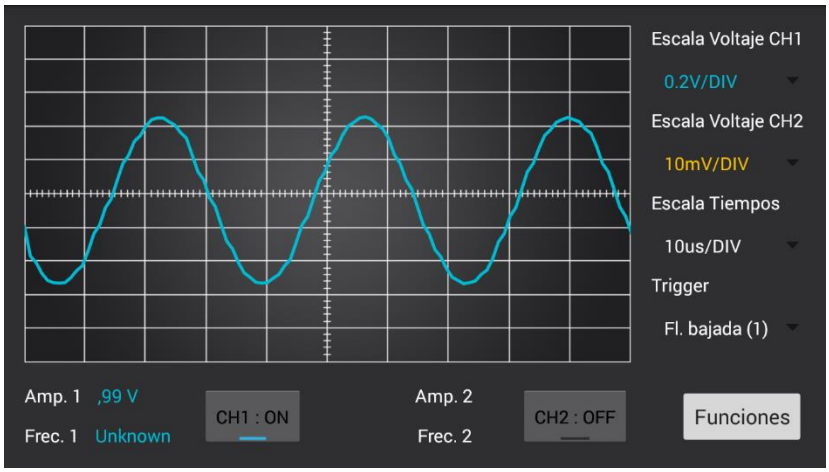



*Figura 6.7. Señal de entrada de prueba 3 de 10 V (escalada) en el osciloscopio del proyecto*

| Equipo | *Tektronix TDS320* | Prototipo desarrollado |
|---|---|---|
| **Amp. Ch1** | 10 V | 0,99 V |
| **Frecuencia** | 20,45 KHz | ~29,41 KHz |

*Tabla 6.3. Resultados en ambos equipos de medida para la prueba 3*

Vemos durante esta prueba que la amplitud de la señal se adapta correctamente al CAD, atenuando su valor por un factor 10 (960 mV a la entrada del CAD), y que después en la visualización la señal tiene un valor correcto (a pesar de la atenuación de la sonda).

La frecuencia nuevamente sigue dando los mismos problemas que en las pruebas anteriores.

**Prueba 4: Señal sinusoidal [1 V, aprox. 100 KHz]**

En esta ocasión se probó a aumentar la frecuencia de la señal de entrada hasta alcanzar las centenas de KHz (100 KHz para esta prueba), mientras que se mantuvo la amplitud en un valor cercano a 1 V.

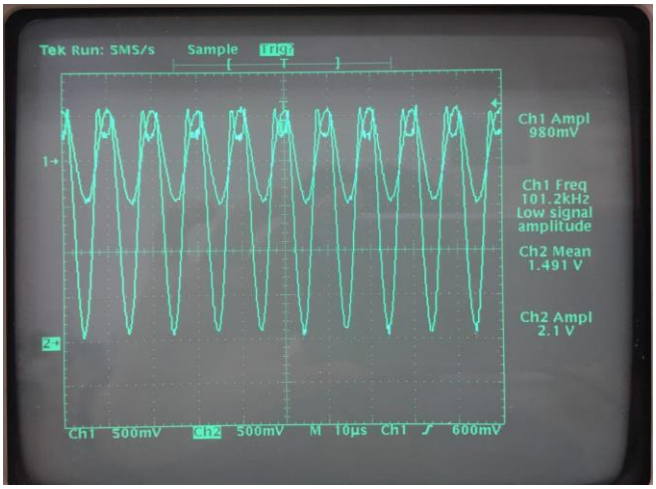



*Figura 6.8. Señal de entrada de prueba 4 (arriba) y señal a la entrada del CAD 1 (abajo) en el osciloscopio comercial*

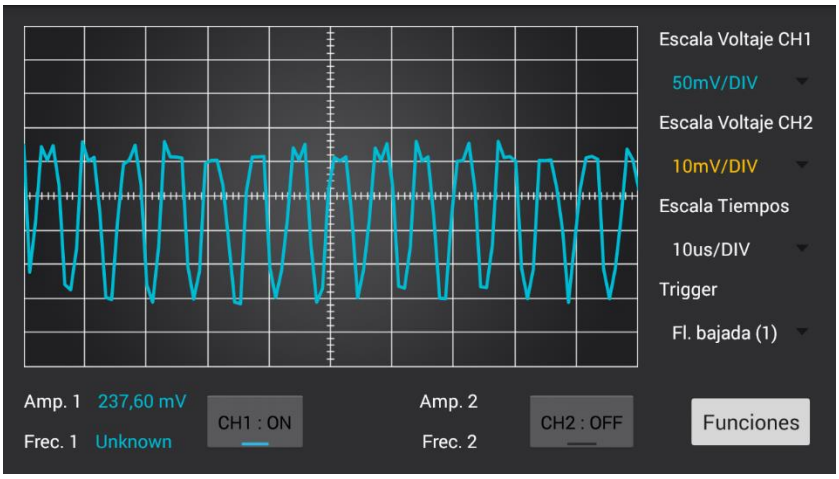



*Figura 6.9. Señal de entrada de prueba 4 de 100 KHz en el osciloscopio del proyecto*

| Equipo | *Tektronix TDS320* | Prototipo desarrollado |
|---|---|---|
| **Amp. Ch1** | 980 mV | 237,60 mV |
| **Frecuencia** | 101,2 KHz | ~142,86 KHz |

*Tabla 6.4. Resultados en ambos equipos de medida para la prueba 4*

Tras esta prueba vemos que la amplitud se ve bastante afectada, de forma que ni siquiera conseguimos una forma de onda adecuada, y su valor (en el caso de no estar escalado por la sonda) sería del doble (~2 V) de la amplitud de la señal de entrada.

La frecuencia, además de no poder ser calculada, estaría también en torno al doble de la de entrada.

**Prueba 5: Señal cuadrada [1 V, aprox. 20 KHz]**

Para la quinta prueba se optó por variar la forma de onda de la señal de entrada. Se eligió una onda cuadrada de amplitud 1 V y frecuencia aproximada de 20 KHz.

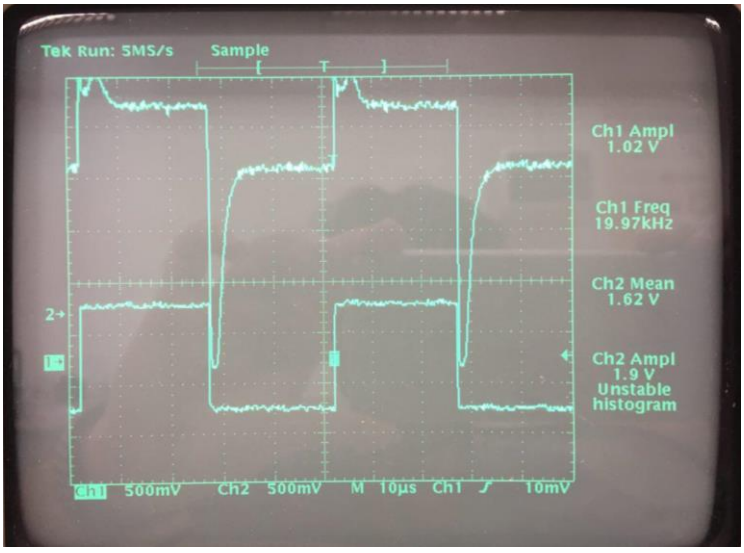



*Figura 6.10. Señal de entrada de prueba 5 (abajo) y señal a la entrada del CAD 1 (arriba) en el osciloscopio comercial*


*Figura 6.11. Señal de entrada de prueba 5 (cuadrada) en el osciloscopio del proyecto*

| Equipo | *Tektronix TDS320* | Prototipo desarrollado |
|---|---|---|
| **Amp. Ch1** | 1,02 V | 242,37 mV |
| **Frecuencia** | 19,97 KHz | ~29,41 KHz |

*Tabla 6.5. Resultados en ambos equipos de medida para la prueba 5*

En esta ocasión, vemos que la señal cuadrada no se visualiza de manera correcta en la aplicación, debido a que a la entrada del CAD (como vemos en la *Figura 6.10*) ya presenta problemas en los flancos de subida y bajada, mostrando ciertos sobre picos tanto positivos como negativos.

La frecuencia como en pruebas anteriores sigue presentando errores.

**Prueba 6: Señal triangular [1 V, aprox. 20 KHz]**

Tras probar con una señal cuadrada se introdujo una señal triangular, manteniendo los valores de amplitud y frecuencia de la prueba anterior.

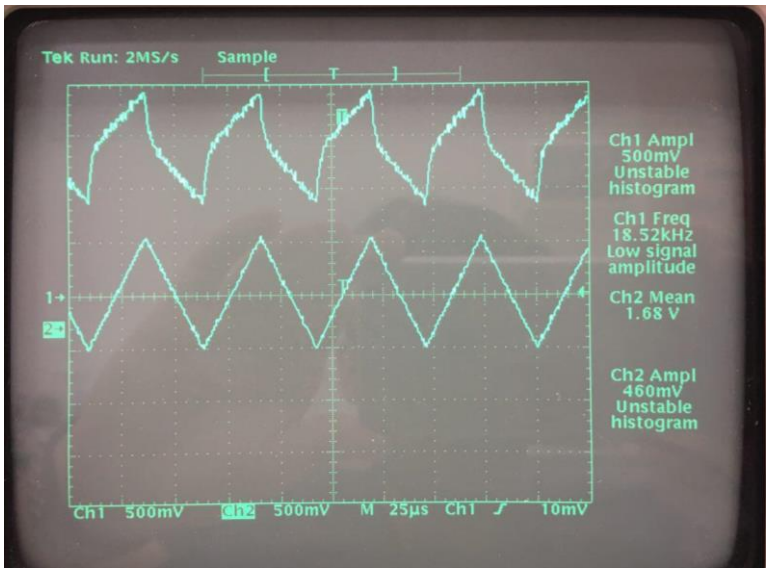



*Figura 6.12. Señal de entrada de prueba 6 (abajo) y señal a la entrada del CAD 1 (arriba) en el osciloscopio comercial*

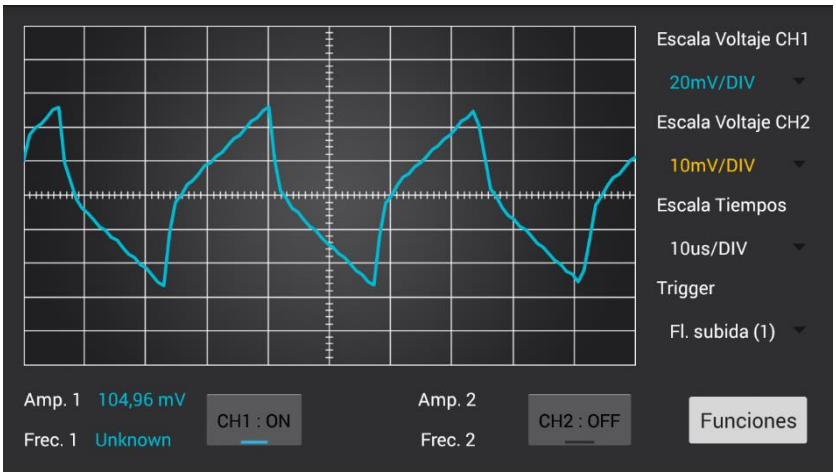



*Figura 6.13. Señal de entrada de prueba 6 (triangular) de 20 KHz en el osciloscopio del proyecto*

| Equipo | *Tektronix TDS320* | Prototipo desarrollado |
|---|---|---|
| **Amp. Ch1** | 1 V | 104,96 mV |
| **Frecuencia** | 18,52 KHz | ~29,41 KHz |

*Tabla 6.6. Resultados en ambos equipos de medida para la prueba 6*

En esta prueba vemos que la forma de onda de la señal también se deforma, ofreciendo una visualización no muy cercana a la realidad. La amplitud, a pesar del escalado, es bastante similar.

La frecuencia, por el contrario, presenta los errores ya comentados en pruebas anteriores.

**Prueba 7: Señal triangular [1 V, aprox. 5 KHz]**

A continuación, realizamos otra prueba con la misma forma de onda, pero reduciendo la frecuencia hasta unos 5 KHz, con el objetivo de ver si el circuito reaccionaba mejor frente a los cambios de pendiente de la señal de entrada.

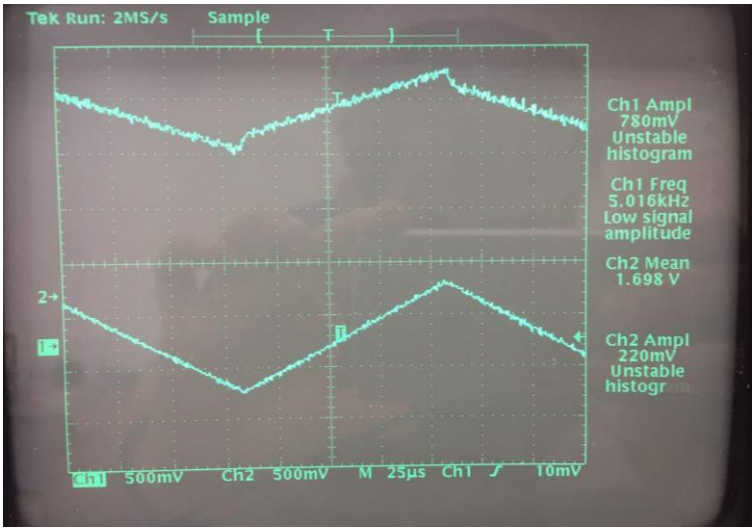



*Figura 6.14. Señal de entrada de prueba 7 (abajo) y señal a la entrada del CAD 1 (arriba) en el osciloscopio comercial*

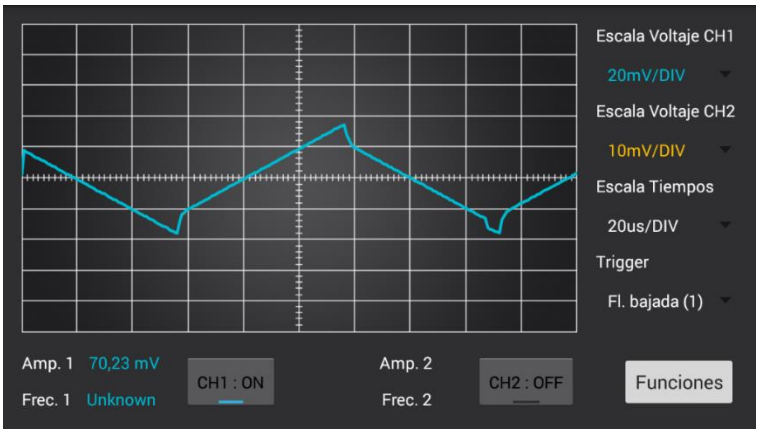



*Figura 6.15. Señal de entrada de prueba 7 (triangular) de 5 KHz en el osciloscopio del proyecto*

| Equipo | *Tektronix TDS320* | Prototipo desarrollado |
|---|---|---|
| **Amp. Ch1** | 780 mV | 70,23 mV |
| **Frecuencia** | 5,02 KHz | ~8,62 KHz |

*Tabla 6.7. Resultados en ambos equipos de medida para la prueba 7*

Tras esta prueba, vemos como la amplitud de la señal (atenuación incluida) es bastante similar a la esperada, mientras que la frecuencia tiene el mismo comportamiento que anteriormente.

**Prueba 8: Canal 1: Señal sinusoidal [1 V, aprox. 20 KHz];**
**Canal 2: Salida OP1 del canal 1;**

Para la última prueba se colocaron dos señales simultáneamente en el prototipo desarrollado, para intentar visualizar el desfase entre ambas señales, haciendo que la señal introducida por el canal 2 fuese la salida del primer amplificador operacional del canal 1, el cual, si recordamos, presenta una configuración inversora. La amplitud y frecuencia fueron de 1 V y 20 KHz aproximadamente.

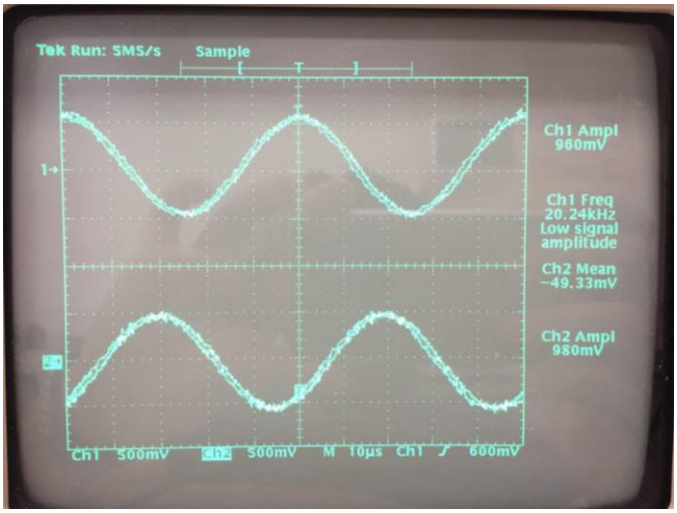



*Figura 6.16. Señal de entrada de prueba 8 (arriba) y señal a la entrada del CAD 1 (abajo) en el osciloscopio comercial*

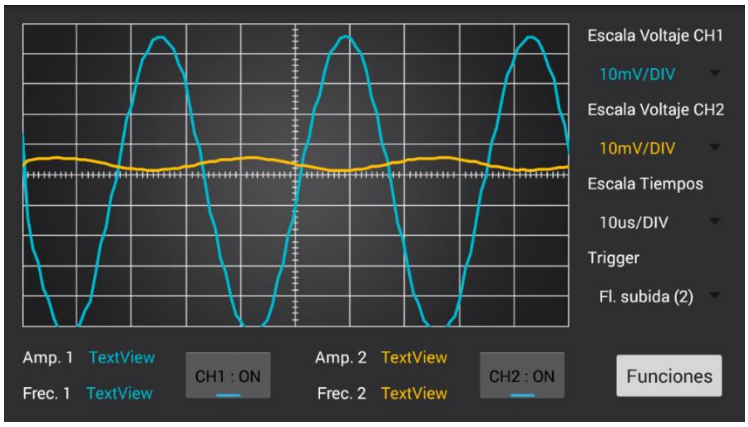



*Figura 6.17. Señales de los canales 1 y 2 en el osciloscopio del proyecto*

| Equipo | *Tektronix TDS320* | Prototipo desarrollado |
|---|---|---|
| **Amp. Ch1** | 960 mV | ~96 mV |
| **Amp. Ch2** | 980 mV | ~0,8 mV |
| **Frecuencia** | 20,24 KHz | ~29,41 KHz |

*Tabla 6.8. Resultados en ambos equipos de medida para la prueba 8*

Para esta última prueba cabe destacar que las sondas utilizadas para medir con el osciloscopio eran ambas sondas x10, mientras que las usadas para introducir las señales en el circuito de la PCB eran una sonda x1 (para el canal 1) y una sonda x10 (para el canal 2), lo que originó que la señal visualizada en el canal 2 estuviese atenuada un factor de 10, y dado de que el interruptor del circuito ya estaba en la escala mínima, no se pudo lograr una visualización más cercana a la del osciloscopio.

Una vez explicado el efecto de atenuación de las sondas, vemos que los valores de amplitud de ambos canales son similares a los esperados.

Vemos también que, en efecto, ambas señales se encuentran en oposición de fase.

En cuanto a la frecuencia, de nuevo presenta errores.

## 6.2. Resultados

### Interferencias

Una de las primeras anomalías que se detectaron al realizar las diferentes pruebas fue que la señal se visualizaba con bastantes interferencias en el dispositivo *Android*, mientras que, en el osciloscopio, exceptuando por el pequeño rizado debido al ruido, la señal se visualizaba de forma correcta. Tras analizar qué podría estar ocurriendo, nos dimos cuenta que a través de los pines utilizados para el conector de programación se introducían interferencias que se transmitían a la señal y hacían que en la visualización apareciesen picos aleatorios que estropeaban algo la visualización de la misma.

**Retardos**

Durante la etapa de diseño y mientras se probaba la aplicación, se notó que la visualización de la señal ofrecía un cierto *lag* (retardo en la visualización) debido a que como la aplicación debe esperar a representar una pantalla para recibir los siguientes datos, el refresco es algo más lento de lo deseado. Aunque esto no es un aspecto demasiado molesto, debería intentar solucionarse en futuras versiones.

**Amplitud medida**

Uno de los aspectos esenciales que destacó durante las pruebas fue la variación en el voltaje de las señales de entrada. Y es que el circuito se comporta de manera diferente, según la frecuencia de la señal de entrada. Podemos ver este efecto en las figuras *Figura 6.1* a *Figura 6.4*, donde observamos que para frecuencias bajas de la señal (hasta unos pocos KHz) sí se efectúa la atenuación de ½ que proporciona el par de resistencias de la etapa de entrada (resistencias de 510 KΩ en *Figura 3.2*), mientras que, para valores más altos (unos 20 KHz), esta atenuación ya no se produce. Este cambio tan variable en la amplitud de la señal no es nada deseado, ya que enmascara el resultado real de las medidas, por lo que se deberá corregir en mejoras futuras.

Los cambios de amplitud (manuales) de la señal fueron satisfactoriamente acondicionados a la entrada del dsPIC, una vez seleccionada la posición correspondiente a la escala que estábamos usando en el interruptor de la PCB, y la representación fue consecuentemente escalada, ofreciendo una visualización cercana a la del osciloscopio comercial. Podemos observar un ejemplo en la *Figura 6.5* y la *Figura 6.6*.

**Frecuencia**

Otro de los aspectos más importantes a tener en cuenta, fue el problema con la medida de la frecuencia. Durante ninguna de las pruebas conseguimos obtener un valor estable de la frecuencia de la señal. Momentáneamente (cuando conseguíamos que dos muestras de la señal coincidiesen en la comparación) se conseguía ver un valor, pero al refrescar la pantalla, este desaparecía. Para evitar este problema, se podría intentar calcular la frecuencia fijándonos sólo en los máximos o mínimos de la señal (indistintamente de que los valores coincidan), pero cuando se intentó de esta manera, los picos de las interferencias falsearon el resultado, haciendo que la aplicación mostrase constantemente números aleatorios y haciendo por tanto imposible una correcta visualización de la misma.

Aparte de este problema, que puede solventarse realizando la cuenta a través de la rejilla de visualización, contamos con un segundo error en la frecuencia. A través de los resultados mostrados en las tablas *Tabla 6.1* a *Tabla 6.8*, vemos que el valor de la frecuencia de las señales representadas en el prototipo y en el osciloscopio varían sensiblemente. Después de estudiar detenidamente este efecto y analizar el flujo de datos capturado y enviado a la aplicación, solamente se entendió una posible causa a este efecto. El problema es debido a que se consideró que la frecuencia de muestreo por parte de la PCB era de 1 MHz (tras configurar los CAD en el código del Firmware), es decir, que se tomaba una muestra cada microsegundo. Sin embargo,

si nos fijamos en los valores, vemos que para cada frecuencia de la señal durante las pruebas teníamos el mismo valor de frecuencia en la aplicación. Esto nos da que pensar que la frecuencia de muestreo efectiva que tenemos no es la calculada teóricamente. Como consecuencia, al intentar representar la señal colocando cada muestra según la tasa de avance calculada en la aplicación (calculada para muestreos a 1 MHz), la señal no se pinta en las coordenadas correctas y la frecuencia que obtenemos es distinta a la real. A la vista de los resultados, y conociendo las frecuencias de entrada exactas gracias al osciloscopio, podemos concretar que la frecuencia de muestreo efectiva de nuestro sistema es de aproximadamente 680 KHz.

Para solucionar el problema de la representación, se debería recalcular el vector de tasa de avance en la aplicación y/o en el Firmware de forma que la señal se dibuje en las coordenadas correctas y la visualización sea correcta.

Tras comprender a que se debían estos problemas con la frecuencia de la señal se tomaron las soluciones oportunas para corregirlos. Como hemos dicho anteriormente, se recalculó el vector de tasa de avance en la app de manera que la representación sea acorde a la realidad (misma frecuencia en cada escala de tiempo en pantalla que la del osciloscopio comercial), y se trató de calcular la frecuencia fijándonos en los máximos y mínimos de la señal, intentando mantener el margen de error debido a los sobre picos lo más bajo posible.

Una vez realizados estos cambios se volvió a comprobar el funcionamiento del sistema introduciendo una señal sinusoidal de frecuencia 20 KHz y 1 V de amplitud.

La *Figura 6.18* y la *Figura 6.19* muestran las señales en ambos equipos (el osciloscopio comercial y el dispositivo *Android*).

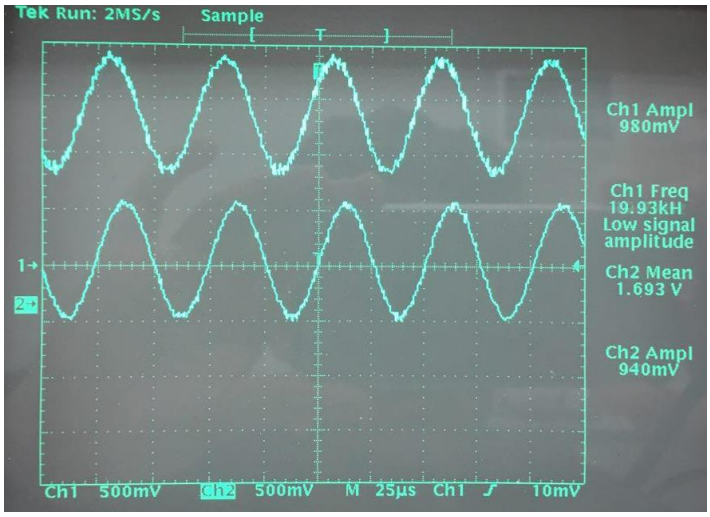



*Figura 6. 18. Señal de entrada de prueba con correcciones (abajo) y señal a la entrada del CAD 1 (arriba) en el osciloscopio comercial*

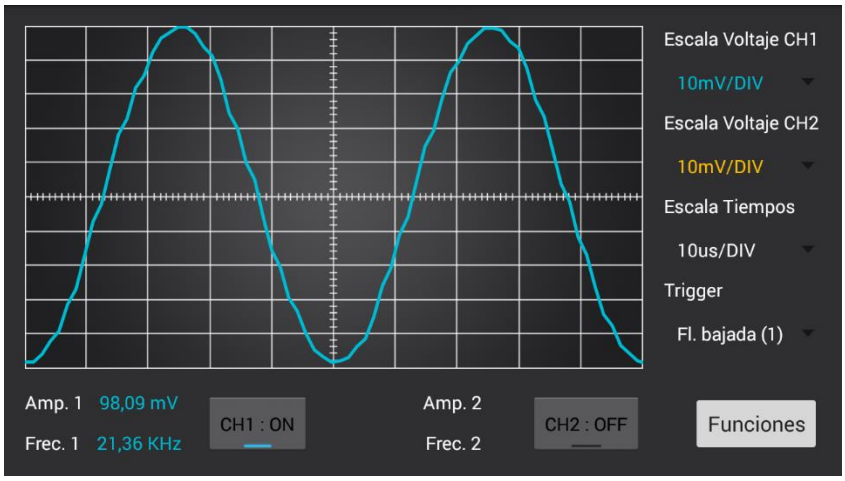



*Figura 6. 19. Señal de entrada de prueba con correcciones en el osciloscopio del proyecto*

Como se puede observar en la captura de la aplicación (*Figura 6.19*), la frecuencia ya puede calcularse y, además, coincide bastante mejor con la frecuencia real de la señal de entrada que nos marca el osciloscopio.

| Equipo | *Tektronix TDS320* | Prototipo desarrollado |
|---|---|---|
| **Amp. Ch1** | 980 mV | 98,09 mV |
| **Frecuencia** | 19,93 KHz | 21,36 KHz |

*Tabla 6. 9. Resultados en ambos equipos de medida para la prueba con correcciones*

**Saturación A.O.**

Durante otra de las pruebas aumentamos la frecuencia de la señal de entrada para acercarnos al punto límite de funcionamiento de nuestro osciloscopio, llegando a observar como a frecuencias más elevadas (cercanas ya a los 100 KHz en adelante) la señal se deformaba en su pico superior (*Figura 6.7*), debido a que la amplitud no es constante con la frecuencia (en realidad, la ganancia) y se disparaba. Al hacerlo, cuando pasa por el operacional se satura, debido a que se le ha sumado a la señal un valor de offset que la eleva por encima de los 3,3 V de margen superior. Para contrarrestar este efecto, y como ya se comentó en la sección relacionada con el Hardware, se debería utilizar un factor de ganancia compensado, para que esta no varíe con la frecuencia.

**Forma de onda**

Al momento de variar la forma de onda de la señal de entrada, nos encontramos con varios resultados adversos. Mientras que el circuito se comporta de manera correcta para señales sinusoidales, no es el caso para señales cuadradas o triangulares. Como podemos ver en la *Figura 6.9* y la *Figura 6.10*, a la hora de introducir una señal cuadrada, el circuito no es capaz de seguir cambios tan bruscos y aparecen unos sobre picos (tanto en el flanco de subida como en el de bajada) que estropean la visualización y el cálculo de la amplitud. Esto ocurre indistintamente de la frecuencia.

Para el caso de la señal triangular ocurre algo parecido, aunque en este caso sí varía con la frecuencia. Cuando introducimos la señal triangular de "alta" frecuencia (*Figura 6.11* y *Figura 6.12*) observamos deformaciones de la señal a la entrada de los CADs, mientras que cuando la frecuencia de la señal de entrada es más baja, estas deformaciones se hacen mucho menores (*Figura 6.13* y *Figura 6.14*). Esto indica que, al igual que con la señal cuadrada, el circuito no es capaz de responder tan rápido a los cambios de pendiente de la señal.

**Dos canales**

Por último, introdujimos dos señales al circuito, para poder observar la visualización de ambas. La única pega es que no se pudo encontrar otra sonda de amplitud 1x, por lo que la señal del canal 2 se vio afectada por la atenuación correspondiente a la sonda de 10x. Lo que se pretendía es poder visualizar la señal de entrada del canal 1 en el propio canal y la esa misma señal una vez pasa por el primer operacional en el canal 2, de manera que se pudiese ver la inversión del operacional (*Figura 6.15* y *Figura 6.16*).

## 6.3. Limitaciones

A la vista de los resultados obtenidos y una vez estudiadas las consecuencias de los mimos, estamos en disposición de analizar las limitaciones encontradas finalmente a nuestro prototipo.

En cuanto a la amplitud de las señales de entrada, el circuito es capaz de adaptar los valores a la entrada del CAD, gracias al mecanismo de escalado a la entrada del mismo, por lo que, en principio, los valores de voltaje de las especificaciones iniciales siguen siendo válidos.

No ocurre lo mismo con las características de ancho de banda, donde, aunque inicialmente la limitación vendría dada por la frecuencia máxima de muestreo del CAD, hemos observado que los diferentes elementos del circuito y la no compensación en frecuencia de alguna de las etapas hace que el ancho de banda efectivo quede reducido a unas decenas de KHz. A pesar de esta limitación el objetivo del proyecto sigue quedando cubierto, ya que es suficiente para realizar prácticas al nivel docente propuesto.

La *Tabla 6.10* resume las características y limitaciones finales de nuestro proyecto.

| | Prototipo desarrollado |
|---|---|
| **N.º canales** | 2 |
| **Nivel *Trigger*** | Auto |
| **Modo disparo** | Flanco subida |
| | Flanco bajada |
| **Tasa de muestreo** | 680 Ksps |
| **Rango de amplitud** | ± 250 V |
| **Ancho de banda** | 50 KHz |
| **Tamaño máx. buffer de datos** | 8 KB |
| **Funciones** | Captura pantalla |
| | Guardado de datos |
| **Conectividad** | USB |

*Tabla 6.10. Especificaciones finales del proyecto*

# Capítulo 7. Costes

En el siguiente capítulo se detallan los costes totales del proyecto para el diseño presentado en el presente documento, analizando cada elemento por separado y teniendo en cuenta los precios tanto de los componentes (en la fecha de realización del trabajo) como de las licencias de los softwares utilizados para el desarrollo del mismo.

Se ha de tener en cuenta que, durante la realización del proyecto, ya se disponía de algunas de las licencias y por lo tanto el coste asociado a los prototipos realizados fue menor.

El objetivo final es definir cuánto costaría replicar el proyecto completamente desde cero, disponiendo, o sin disponer, de los archivos de código y diseño de los componentes.

Por último, cabe destacar que uno de los puntos fuertes de este proyecto es que se basa en poder utilizar elementos como *Smartphones* o *Tablets* para la representación y control de las señales y se asume que, en la actualidad, la mayoría de los usuarios disponen ya de estos dispositivos, de modo que el coste total del proyecto se abarata gracias a no tener que adquirir este tipo de dispositivos.

## 7.1. Coste de diseño y fabricación

Como ya hemos comentado en capítulos anteriores, la fabricación de la PCB se llevó a cabo a través de la empresa *PCBWay*, a la cual se le enviaron los ficheros de diseño para que, en el plazo de una semana, nos enviaron las placas ya fabricadas.

El coste para el tipo de placa solicitada asciende a 0,5$ (aprox. 0,45€), pero dado que el pedido mínimo que se realizó fue de 10 PCBs el precio se calcula en 5$ (aprox. 4,46€).

Además, deben añadirse los gastos de envío, que en nuestro caso ascendían a unos 23€.

Con todo esto el total para la fabricación de nuestro hardware se estima en unos **27,64€**.

## 7.2. Coste de componentes

Una vez realizado el diseño completo de nuestro hardware, el programa utilizado para ello nos permite disponer de una lista total de materiales (*Bill Of Materials)*, la cual utilizaremos para realizar los cálculos siguientes y puede consultarse en el Anexo C.

A pesar de que en el laboratorio ya se disponía de algunos de los componentes necesarios y algunos otros con los que podíamos sustituir los inicialmente planteados, el análisis económico para esta sección incluye el coste total para todos ellos.

Una vez sumado el coste de todos los componentes obtenemos un total de **32,81€**.

En nuestro caso, ya que no se pudo localizar el potenciómetro y finalmente fue sustituido por una resistencia, el coste total se redujo a **30,04€**.

## 7.3. Coste de licencias

La mayoría de los softwares utilizados disponen de versiones gratuitas, aunque con limitaciones de funciones. A pesar de ello, estas versiones gratuitas son más que suficientes para realizar las tareas necesarias en el diseño y programación de los sistemas del proyecto.

No es el caso del software utilizado para el diseño del hardware, *Proteus*, el cual únicamente dispone gratuitamente de una versión de prueba por tiempo limitado. Aunque existen numerosas distribuciones de este programa, con diferentes funciones cada una, la licencia más básica (*Proteus PCB Design Starter Kit*) nos serviría para nuestro propósito de diseño. Esta licencia tiene un precio de **200,00€**.

## 7.4. Presupuesto final

Una vez analizados los costes de cada elemento por separado, podemos realizar una estimación del coste total del proyecto. En este caso, como existía un número mínimo de unidades en la compra de algunos componentes se expondrán dos presupuestos distintos, uno para la fabricación de un único prototipo una vez estemos en la etapa de producción, y otro referente al coste para el primer prototipo, donde habríamos tenido que hacer el pedido mínimo de cada elemento.

| Costes | Primer Prototipo | Precio por PCB |
|---|---|---|
| **Fabricación PCB** | 4,46€ | 0,45€ |
| **Envío PCB** | 23,18€ | 0€ (Sólo un envío) |
| **Componentes** | 53,25€ | 32,81€ |
| **Total** | 80,89 € | 33,26€ |
| **Licencias** | 200€ | 0€ (Sólo la primera vez) |
| **Total, con licencias** | 280,89€ | 33,26€ |

*Tabla 7.1. Presupuesto final por unidad y para el primer prototipo*

## 7.5. Comparativa con otros equipos del mercado

A la vista de los resultados obtenidos, podemos realizar una pequeña comparativa con alguno de los osciloscopios comerciales que encontramos en el mercado.

Si bien es cierto que muchos de los osciloscopios comerciales poseen características superiores a las de nuestro proyecto, si se quisiese comprar un osciloscopio para un laboratorio del nivel que nosotros estudiamos las características mínimas que debiese tener serían las mismas. El número de canales debería ser de mínimo dos canales, las características de amplitud de entrada similares a las del proyecto y un rango de frecuencias de entrada también similar (aunque a menudo este valor será mayor que el rango alcanzado en nuestro proyecto).

Para la comparativa veremos dos osciloscopios comerciales con diferentes prestaciones intentando contrastar equipos más asequibles y de altas prestaciones.

- Hantek DSO5102P, **259,98€**

Se trata de un osciloscopio de dos canales, con un ancho de banda de 100 MHz, muestreo a 1 Gsps y 8 bits de resolución vertical.


*Figura 7.1. Osciloscopio digital Hantek DSO5102P*

- Siglent SDS1104X-E, **520,00€**

Se trata de un osciloscopio de cuatro canales, con ancho de banda de 100MHz, muestreo a 1 Gsps y una resolución vertical de 10 bits. Además, incluye funciones para decodificación de protocolos y multitud de selección de disparos.

*Figura 7.2. Oscilosopio digital Siglent SDS1104X-E*

Por todo lo expuesto anteriormente el osciloscopio diseñado y construido en este trabajo es una muy buena opción, altamente más económica que sus alternativas, lo cual encaja a la perfección con los objetivos del proyecto.

# Capítulo 8. Conclusiones

Durante el presente proyecto hemos descrito la fabricación de un osciloscopio digital basado en *Android* desde la etapa de diseño hasta la de fabricación, construcción, montaje y puesta en marcha del mismo.

Este trabajo tenía como objetivo principal dotar a los centros de enseñanza de alternativas para la medida de señales más económicas que los equipos comerciales existentes para posibilitar la realización de prácticas básicas en laboratorio. A la vista de los resultados obtenidos, podemos concretar que el objetivo ha sido cumplido, a pesar de que siempre hay margen para mejoras.

El Trabajo de Fin de Grado es la culminación académica al proceso de aprendizaje que un estudiante adquiere durante sus estudios de Grado y por lo tanto debe demostrarse que el estudiante ha adquirido los conocimientos y competencias propios del mismo, como las capacidades para trabajar en entornos de laboratorio, desarrollar metodologías y destrezas de aprendizaje autónomo eficiente o redactar, desarrollar y firmar proyectos en el ámbito de la ingeniería de telecomunicación.

En concreto, para la Mención en Sistemas Electrónicos, durante la realización de este proyecto se han logrado además las siguientes competencias específicas:

- Capacidad para construir, explotar y gestionar sistemas de captación, transporte, representación, procesado, almacenamiento, gestión y presentación de información multimedia, desde el punto de vista de los sistemas electrónicos.

- Capacidad de realizar la especificación, implementación, documentación y puesta a punto de equipos y sistemas electrónicos considerando todos los aspectos técnicos.

- Capacidad para aplicar la electrónica como tecnología de soporte en otros campos y actividades.

- Capacidad de diseñar circuitos de electrónica analógica y digital, de conversión analógico-digital, de alimentación y conversión de energía eléctrica para aplicaciones de telecomunicaciones y las Tecnologías de la Información y las Comunicaciones.

- Capacidad para diseñar dispositivos de interfaz, captura de datos y almacenamiento, y terminales para servicios y sistemas de telecomunicación.

- Capacidad para utilizar instrumentación electrónica y sistemas de medida, y analizar y solucionar problemas de interferencias y compatibilidad electromagnética.

Todas ellas, incluidas y descritas en el marco de la guía docente para la realización del Trabajo de Fin de Grado.

Cabe destacar, además, que, durante la realización del proyecto se han adquirido y desarrollado nuevos conocimientos como la programación de aplicaciones para *Android*, con las que se tenía muy vaga experiencia, la interpretación y uso de la programación en lenguaje *Java* o la captura de datos y su envío a través de ciertos protocolos. Ninguna de estas capacidades ha sido estudiada a lo largo de la formación académica, bien por no haber cursado las asignaturas optativas correspondientes, o por no ser, de hecho, competencias directas del Grado estudiado.

La realización de este trabajo a supuesto, por tanto, no solo una demostración de la destreza adquirida, sino también una gran satisfacción personal al ver que se pueden resolver problemas, dudas y adversidades de manera autónoma y eficiente.

## 8.1. Trabajo futuro/mejoras

Debido a que a menudo las especificaciones de los proyectos en ingeniería cambian a lo largo del propio proyecto, continuamente se están realizando cambios y mejoras al sistema. Inicialmente, el proyecto de estudio comenzó implementando un sistema totalmente diferente (como se analizó durante las prácticas en empresa) que fue cambiando para adecuarse a nuevas especificaciones o cumplir ciertos objetivos. No obstante, el sistema desarrollado finalmente no está para nada concluso, sino que puede seguir desarrollándose e implementando nuevas funciones y mejoras. Algunas de las cuales se comentan a continuación.

- Dado que el escalado de las señales de entrada tiene que hacerse de manera manual, desplazando el interruptor físico situado en la PCB, sería interesante que esto pudiese hacerse de manera automática una vez el usuario seleccionase la escala preferida en la aplicación. Esto podría hacerse colocando unos interruptores analógicos que fuesen controlados (habilitados) mediante salidas digitales libres del dsPIC.

- Aunque la alimentación no ha supuesto un problema y ya que el consumo del circuito es bastante reducido podría ser útil disponer de una batería para dotar de mayor movilidad y portabilidad al sistema.

- Puesto que algunas de las huellas de los componentes resultaron erróneas o defectuosas, se debería realizar una nueva versión de la PCB donde esto se corrigiese. Así mimo, algunos de los componentes utilizados son difíciles de conseguir o bien están obsoletos, por lo que en esta nueva versión deberían de sustituirse por componentes equivalentes. Todos estos elementos se encuentran resumidos en la *Tabla 3.3*.

- A pesar de que el uso que se le dará al sistema lo sitúa en el marco de prácticas docentes, sería útil añadir un fusible a la entrada del circuito, como elemento de protección contra corrientes elevadas.

- Como hemos visto durante las pruebas, la frecuencia de la señal influye de manera directa en la forma y amplitud de la misma, por lo que se deberían colocar condensadores para compensar este efecto en las etapas donde mencionamos que este efecto ocurre.

- Actualmente la placa de circuito se manipula directamente de forma que el usuario está en contacto con los componentes y conectores. Para evitar que puedan producirse desconexiones accidentales o descargas electrostáticas en los componentes, podría diseñarse una carcasa que protegiese la PCB y solo dejase accesibles los controles necesarios.

- En cuanto al Software y Firmware existen distintas mejoras disponibles, como incluir nuevas funciones a realizar con las señales, incluir un selector de nivel manual de *trigger*, realizar un muestreo en tiempo equivalente para poder medir señales de mayor frecuencia, o utilizar la clase *SurfaceView* para trabajar con *OpenGL* y obtener mejores resultados a la hora de representar las señales, a la vez que reducimos el *lag* que se comentó durante las pruebas.

- Por último, aunque supone un esfuerzo mucho mayor, ya que se tendría que cambiar prácticamente todo el circuito realizado, la utilización de una *FPGA* como núcleo central del sistema (en lugar del DSP utilizado), dotaría a este de una mayor velocidad, lo cual para el propósito del equipo de medida que realizamos sería una ventaja muy grande.

## 8.2. Recomendaciones

Pese a que el funcionamiento del osciloscopio de este proyecto es fácil e intuitivo, deben recordarse al usuario algunos aspectos importantes para la seguridad y el buen uso de los equipos, de forma que se alargue el tiempo de vida de las unidades y el usuario mantenga una buena rutina de operación que le ayude a lograr un buen desempeño del equipo.

- Dado que el escalamiento no se realiza de manera automática en el Hardware, antes de introducir una señal en el osciloscopio, el usuario debe fijarse en que el interruptor selector de ganancia este colocado en la posición correspondiente a la escala de voltajes en la que se encuentre la señal de entrada. Recordemos que esto viene reflejado en la *Tabla 3.1*.

- Debe tenerse en cuenta el tipo de sonda que se utilice para conectar las señales al circuito, de modo que se analice correctamente el resultado obtenido en la visualización. Debe recordarse que los valores de amplitud de la señal representada están escalados según la atenuación que introduzca la sonda, así como el factor de ganancia de la PCB.

- Igualmente se debe tener en cuenta que, a frecuencias bajas, aparecerá el efecto de atenuación por ½, el cual puede enmascarar el resultado final si no se recuerda.

- Cabe recordar también al usuario que se debe seleccionar una escala de voltajes en la aplicación acorde a la posición del interruptor de ganancia, como marca la *Tabla 5.1*. En caso contrario obtendremos resultados erróneos, al igual que en los casos anteriores.

- Por último, se recomienda a los usuarios que no excedan las limitaciones de voltaje para las que está diseñado el proyecto, para evitar así cualquier desperfecto en los sistemas y elementos que lo componen.

# Bibliografía


*[1]* Aslam, Husnain. (2017): *"USB Communication using Android Accessory Kit"* [en línea] Disponible en http://www.devprofessor.com/android/usb-communication-using-android-accessory-kit/

*[2]* Bejarano Bueno, J.M. (2015): *Trabajo Fin de Grado - Datalogger con enlace Bluetooth a sistema Android*. Escuela Politécnica Superior de Jaén, Universidad de Jaén

*[3]* BitScope Designs. *"BitScope Circuit Design"* [en línea] Disponible en http://www.bitscope.com/design/hardware/

*[4]* Borrego Colomer, A. *"Tutorial del Osciloscopio"* [en línea] Disponible en www.ing.unp.edu.ar

*[5]* Dibujes Salgado, C. M., E. A. Torres Morales (2008): Diseño e implementación de un osciloscopio con un dsPIC - Proyecto previo a la obtención del título de ingeniero en electrónica y control. Facultad de Ingeniería Eléctrica y Electrónica, Escuela Politécnica Nacional de Quito

*[6]* Di Cerbo, M., A. Rudolph (2010): *Using Android in Industrial Automation – Technical Report*. University of Applied Sciences Northwestern Switzerland for the Institute of Automation

*[7]* Google Developers. *"Actividades"* [en línea] Disponible en https://developer.android.com/guide/components/activities.html?hl=ES

*[8]* Google Developers. *"Android Open Accessory (AOA)"* [en línea] Disponible en https://source.android.com/devices/accessories/protocol.html

*[9]* Google Developers. *"Custom Accessories"* [en línea] Disponible en https://source.android.com/devices/accessories/custom

*[10]* Google Developers. *"USB accessory overview"* [en línea] Disponible en https://developer.android.com/guide/topics/connectivity/usb/accessory.html#top_of_page

*[11]* Google Developers. *"USB host and accessory overview"* [en línea] Disponible en https://developer.android.com/guide/topics/connectivity/usb/

*[12]* Hackeando el Genoma (2014): *"Conectar Android con Arduino por USB"* [en línea] Disponible en https://hackeandoelgenoma.com/2014/08/conectar-android-con-arduino-por-usb/

*[13]* Hernández Mangas, Jesús M. (2015): *Apuntes Protocolo USB*. Interconexión de Sistemas Digitales

*[14]* Hoffman, Pablo, M. Szmulewicz (2011): *"Osciloscopio por USB de 40MHz"*, Saber Electrónica, 265 pp. 57-61

*[15]* López, Pedro: *Tema 1. Osciloscopios*. Equipos Electrónicos de Medida y Alimentación

*[16]* Microchip Technology (2011): Android Accessory Development Board – PIC24F_ADK (Schematic) rev. 3. Disponible en http://ww1.microchip.com/downloads/en/DeviceDoc/PIC24F_ADK_Schematic.pdf

*[17]* Microchip Technology (2013): *dsPIC33/PIC24 Family Reference Manual - DS70005131A - Oscillator Module.* Disponible en http://ww1.microchip.com/downloads/en/devicedoc/70005131a.pdf

*[18]* Microchip Technology (2010): dsPIC33/PIC24 Family Reference Manual - DS70571B - Section 25. USB On-The-Go (OTG). Disponible en http://ww1.microchip.com/downloads/en/DeviceDoc/S25.pdf

*[19]* Microchip Technology (2010): dsPIC33/PIC24 Family Reference Manual - DS70608B - Section 24. Programming and Diagnostics. Disponible en http://ww1.microchip.com/downloads/en/devicedoc/s24.pdf

*[20]* Microchip Technology (2012): *dsPIC33/PIC24 Family Reference Manual - DS70621C - Section 16. Analog-to-Digital Converter (ADC)*. Disponible en http://ww1.microchip.com/downloads/en/devicedoc/70621c.pdf

*[21]* Microchip Technology (2012): *dsPIC33EPXXX(GP/MC/MU)806/810/814 and PIC24EPXXX(GP/GU)810/814 Datasheet*. Disponible en http://ww1.microchip.com/downloads/en/DeviceDoc/70616g.pdf

*[22]* Microchip Technology. *"Hardware Tools"* [en línea] Disponible en https://microchipdeveloper.com/hwtools:start

*[23]* Microchip Technology. *"Microchip Libraries for Applications"* [en línea] Disponible en https://www.microchip.com/mplab/microchip-libraries-for-applications

*[24]* Microchip Technology. *"MPLAB® X Integrated Development Environment (IDE)"* [en línea] Disponible en https://www.microchip.com/mplab/mplab-x-ide

*[25]* Microchip Technology. *"MPLAB® XC16 C Compiler"* [en línea] Disponible en https://microchipdeveloper.com/xc16:start

*[26]* Microchip Technology. (2012): *"PIC24F Accessory Development Starter Kit for Android"* [en línea] Disponible en https://www.microchip.com/Developmenttools/ProductDetails/DM240415

*[27]* Microchip Technology. *"PICkit™ 3 (PG164130)"* [en línea] Disponible en https://microchipdeveloper.com/pickit3:start

*[28]* Microchip Technology. *"Software Development Tools for Microchip MCUs"* [en línea] Disponible en https://microchipdeveloper.com/swtools:start

*[29]* Microchip Technology. *"Universal Serial Bus (USB)"* [en línea] Disponible en https://microchipdeveloper.com/usb:start

*[30]* Microchip Technology. *"USB Transfer Types"* [en línea] Disponible en http://microchipdeveloper.com/usb:transfer

*[31]* PCBWay *"Capacidades de PCB"* [en línea] Disponible en https://www.pcbway.es/capabilities.html

*[32]* Pérez García, M.A., Otros (2004): *Instrumentación Electrónica*. Thomson/Paraninfo, pp. 591–616

*[33]* *"Analogue Oscilloscope: cathode ray oscilloscope"* [en línea] Disponible en http://www.electronics-notes.com/articles/test-methods/oscilloscope/analogue-oscilloscope.php

*[34]* *"Android + Arduino = 🖤"* (2011) [en línea] Disponible en https://habr.com/en/post/123361/

*[35]* *"Android + PIC Microcontroller"* [en línea] Disponible en https://stackoverflow.com/questions/5054626/android-pic-microcontroler

*[36]* *"Android bluetooth Arduino data receive"* [en línea] Disponible en http://www.electronoobs.com/eng_arduino_tut20_2.php

*[37]* *"Arduino Android USB Serial Communication With OTG Cable"* [en línea] Disponible en https://www.instructables.com/id/Arduin-Adroid-USB-Serial-Communication/

*[38]* *"Document Library"* [en línea] Disponible en http://www.usb.org/developers/docs/

*[39]* *"FTDI anuncia un integrado host USB optimizado para plataformas Android"* (2012) [en línea] Disponible en http://electronica-pic.blogspot.com/2012/08/ftdi-anuncia-un-integrado-host-usb.html

*[40]* *"How to use canvas in your android apps - Part 1"* [en línea] Disponible en http://www.helloandroid.com/tutorials/how-use-canvas-your-android-apps-part-1

*[41]* *"Mandar datos por usb a tablet android con OTG"* (2013) [en línea] Disponible en http://www.todopic.com.ar/foros/index.php?topic=40418.0

*[42] "Osciloscopio"* [en línea] Disponible en http://es.wikipedia.org/wiki/Osciloscopio

*[43] "Osciloscopio Android casero"* [en línea] Disponible en http://soloelectronicos.com/tag/osciloscopio-bluetooth-casero/

# Anexos

## A. Memoria de prácticas en empresa

En este anexo se incluye la memoria de prácticas en empresa elaborada previo a la realización del Trabajo de Fin de Grado y que concretó cuales serían las especificaciones previas del proyecto, así como los objetivos del mismo.

# 1. Datos generales de la práctica

Código de la práctica: 16/512-024C
Tipo de práctica (curricular o extracurricular): Curricular
Tutor académico: Iván Santos Tejido

**Datos personales del estudiante**

Apellidos y nombre: Ortiz de Latierro Delgado, David
DNI: 71710341K
Dirección: C/ Antigua, 4, 1º
Localidad: Toro (Zamora)
Teléfono: 637163183
Correo electrónico: david.ortiz-latierro@alumnos.uva.es

**Datos de la empresa**

Razón social: Universidad de Valladolid
Centro de trabajo: ETSI de Telecomunicación, biblioteca del departamento de Electrónica, Parque Científico UVa, GIR de Nuevas Tecnologías para la mejora de la calidad de la enseñanza
Dirección: Universidad de Valladolid
Localidad: Valladolid

Tutor de la entidad: Miguel Ángel González Rebollo
Cargo en la entidad: Tutor Empresa
Teléfono: 983184956(ext. 4956)
Correo electrónico: mrebollo@eii.uva.es

**Calendario de la práctica**

Fecha de inicio: 07/11/2016
Fecha de fin: 03/02/2017
Horario: 9:00 – 13:00
Total horas realizadas: 150

## 2. Descripción de la entidad de acogida (breve historia, ubicación, sector y actividades de la entidad, así como otros datos que considere relevantes)

La entidad en la que he realizado las prácticas es el GIR de Nuevas Tecnologías para la mejora de la calidad de la enseñanza.

### Grupo de Innovación Docente TIA

TIA (Tecnología, Innovación y Aprendizaje), es un grupo multidisciplinar creado en 2014, formado por profesores de la UVa de diferentes Centros (E.I. Informática., F. Ciencias, E.I. Industrial) y Departamentos (Informática, Física Aplicada, Física de la Materia Condensada). Además, colabora habitualmente con otros profesores de distintas disciplinas.

### Objetivos del Grupo

El trabajo experimental en ciencia y tecnología es una herramienta clave para el aprendizaje. Por ello los laboratorios de prácticas resultan imprescindibles para el aprendizaje de estas disciplinas. Sin embargo, no siempre se dispone de las instalaciones o los medios adecuados para que los estudiantes puedan "aprender haciendo". El objetivo fundamental del grupo está centrado en estudiar como las TICs (Tecnologías de la Información y la Comunicación) pueden facilitar y hacer accesible al mayor número posible de estudiantes esta estrategia basada en el aprendizaje práctico.

### Líneas de investigación en curso

En estos momentos el Grupo TIA centra sus esfuerzos en el desarrollo de entornos y herramientas de bajo coste que permitan el diseño de prácticas docentes, fundamentalmente de física, tanto en entornos formales (Laboratorios de prácticas), como en entornos informales. Para ello se centra principalmente en la utilización tanto de sensores y electrónica (*Raspberry Pi, Arduino*), como de los dispositivos móviles (Smartphones, Tablets, wearables).

Actualmente cuenta con una dilatada experiencia en este campo y numerosas publicaciones en revistas y congresos tanto nacionales como internacionales.

# 3. Descripción concreta y detallada de las tareas, trabajos desarrollados y departamentos de la entidad a los que ha estado asignado.

## Resumen

Durante las prácticas en empresa, se estudiaron las diferentes alternativas y requisitos para desarrollar una aplicación de osciloscopio digital en el Sistema Operativo Android.

El plan de trabajo se dividió en cuatro fases diferenciadas que se detallan a continuación:

- Documentación: En la fase de documentación se estudiaron algunas de las soluciones ya existentes desarrolladas, o en desarrollo, de osciloscopios digitales en Smartphones y Tablets, basadas en Android. Para ello se realizó una búsqueda de información sobre varias alternativas reparando en distintos aspectos fundamentales como son: el medio de adquisición de datos, el número de canales disponibles y como se consiguen, resolución máxima y rango de medidas, sistemas de protección contra sobrecargas, medidas que puede realizar, o fabricación de una sonda (en el caso de no utilizar una comercial).
  Se puso también interés en la forma de la interfaz de control del osciloscopio, para así poder obtener ideas a la hora de realizar una propia.
- Caracterización del hardware disponible: En la segunda fase se analizaron las especificaciones del Smartphone disponible para pruebas, con el objetivo de caracterizar sus principales componentes y averiguar cuáles eran sus ventajas y limitaciones a la hora de su uso con el osciloscopio.
- Acceso a la toma de audio: En esta tercera fase se investigó como es el acceso a la toma de audio de los dispositivos Android, y como podemos utilizar este conector como entrada de datos para la adquisición de las señales y su posterior tratamiento.
- Planteamiento de la opción más adecuada: Para la última fase, una vez vistas las diferentes soluciones existentes y analizados los medios de los que disponemos, planteamos cual sería la opción más adecuada a desarrollar en el posterior Trabajo Fin de Grado, eligiendo las funciones y características más relevantes a incluir en nuestra aplicación.

## Objetivos

El objetivo principal de las prácticas en empresa es analizar distintas alternativas ya existentes de soluciones para el desarrollo de un osciloscopio digital en Android, poniendo especial interés en averiguar cuáles son sus ventajas y limitaciones en cuanto a sus características y especificaciones y sus principales funciones.
A continuación, se realizará un estudio del hardware del que disponemos, para comprobar qué funciones podremos realizar con sus características técnicas y cuáles son las limitaciones que nos presenta.
Después, tocaremos el tema de la programación en Android, centrándonos en la captación de las señales a través de la toma de audio de los dispositivos como vía de entada.

Por último, una vez exploradas las diferentes aplicaciones ya resueltas y teniendo en cuenta el hardware disponible, buscaremos la solución que mejor se adecue a nuestro caso concreto, lo cual sentará las bases para la realización del posterior Trabajo Fin de Grado, que contendrá el desarrollo y la realización del osciloscopio digital.

## ¿Qué es un osciloscopio?

Un osciloscopio es un instrumento utilizado para visualizar, medir y analizar tensiones variables en el tiempo. Podemos distinguir dos tipos:

- Osciloscopios analógicos (CRO, *"Cathode Ray Oscilloscope"*).
- Osciloscopios digitales (DSO, *"Digital Storage Oscilloscope"*).

En concreto nos vamos a centrar en los digitales, cuyas características principales son las siguientes:

- Muestrean la señal de entrada y almacenan la información hasta tener un número de puntos suficientes para representarla.
- La representación no es en tiempo real.
- La velocidad de actualización de la pantalla está limitada por el procesamiento de las muestras.
- Son adecuados para visualizar señales de baja frecuencia, transitorios y la parte de la señal anterior al punto de disparo.

## Alternativas existentes de osciloscopios digitales

Primeramente, estudiaremos algunas de las soluciones comerciales que existen en el mercado, analizando sus principales características y funcionalidades.

### *Osciloscopio Imex Micsig Tablet*


***Figura 1. Osciloscopio Micsig tBook T02041***

Este osciloscopio tiene formato de Tablet, lo cual le dota de gran portabilidad y comodidad de uso. Además, cuenta con un ancho de banda de hasta 1GHz y una tasa de muestreo de hasta 5GS/s (Giga muestras por segundo). Según la web del fabricante está orientado a

cubrir aplicaciones de comunicaciones, semiconductores, informática, instrumentación e investigación/docencia entre otras.
Su pantalla es una LCD táctil de 10,1" con una resolución de hasta 1024*600px.
Entre sus principales características destacan:

- Ancho de banda de hasta 1GHz
- Tasa de muestreo de 1GS/s – 5GS/s
- Dos canales o hasta 4 aislados
- Tiempos de subida < 5ns
- Acoplo de entrada DC, AC y GND
- Resolución vertical de 8bits, entre 2mV/div y 5V/div
- Resolución horizontal de 200ps/div a 1000s/div
- Modo de disparo: flanco, pulso, lógico, etc.
- Modo de decodificación de protocolos: UART, LIN, CAN, SPI, I2C, 1553B, 429
- Funciones matemáticas: +, -, *, / y FFT
- Sonda: x10, 13pF, 10MΩ mediante BNC


***Figura 2. Interfaz de usuario en el tBook T02041***

Además de estas funciones, el equipo puede utilizarse también como generador de señales, multímetro y grabador de señales, gracias a su memoria interna. Por último comentar que tiene una autonomía de unas 8 horas continuadas y que también soporta conectividad WIFI, LAN y USB 2.0. Su precio es de unos 875,00€.

### *DSO NANO V3*

Este es un osciloscopio digital de bolsillo, de un tamaño similar al de un Smartphone, aunque con algo más de grosor, muy fácil de operar.


***Figura 3. Osciloscopio DSO NANO v3 (Accesorios incluidos en la izquierda)***

Está basado en un microcontrolador de 32bits ARM Cortex-M3 y equipado con una pantalla a color de 320*240px no táctil. Lleva un puerto USB para conectarse con el ordenador y cargar su batería. Sus aplicaciones principales son la docencia, reparaciones eléctricas e ingeniería. Sus características más importantes son:

- Ancho de banda de hasta 200KHz
- Tasa de muestreo de 1MS/s (12bits)
- Un canal
- Acoplo de entrada DC
- Resolución vertical entre 0,5V/div y 10V/div
- Resolución horizontal de 1us/div a 10s/div
- Modo de disparo: 6 (Auto, Normal, Simple, Ninguno, Escaneo y Ajuste)
- Sonda: x10, 500KΩ mediante jack de audio

Este otro osciloscopio también puede utilizarse como generador de señales gracias a una tarjeta de memoria externa donde podemos grabar nuestras señales, aunque son muy limitadas. Su precio es de unos 90,00€.

### *LabNation SmartScope*

En este caso, el dispositivo también realiza funciones extras a la del osciloscopio, como son la de analizador lógico y generador de ondas. El *SmartScope* permite que cualquier persona posea un laboratorio personal de alta tecnología tan solo conectándolo a un PC, portátil, Tablet o Smartphone.

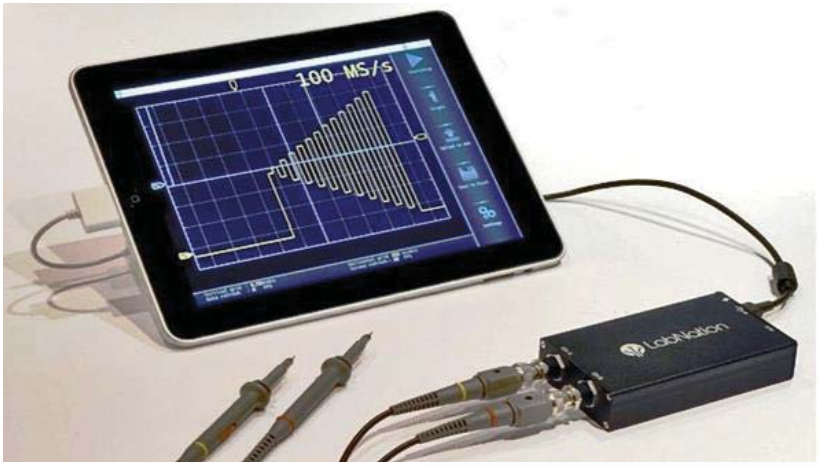



***Figura 4. SmartScope conectado a iPad mediante USB***

A pesar de analizarlo como un dispositivo comercial, lo único que adquirimos es la placa controladora y las sondas, ya que el proyecto es de código abierto y todas las APIs están a disposición del usuario, tanto para Windows como para Linux, MacOS, Android o iOS, así como el acceso completo a su FPGA que lo hace funcionar.

Por lo tanto, necesitamos un terminal al cual conectarlo, mediante USB. Como buscamos soluciones basadas en Android, nos centramos en su conexión con los Smartphones y Tablets.

En este caso, la conexión es como hemos mencionado mediante USB (USB-OTG, "On The Go"), una funcionalidad requerida pero que se está convirtiendo en un estándar para dispositivos Android, donde es compatible desde Android v3.1, aunque depende de la antigüedad y la calidad del Smartphone el que éste sea compatible con el estándar.

Aunque vemos que tiene varias aplicaciones, nos centraremos en analizar sus características como osciloscopio:

- Ancho de banda de 30MHz (En el punto de -3dB)
- Tasa de muestreo de 100MS/s (por canal)
- Dos canales
- Acoplo de entrada DC, AC y GND
- Resolución vertical de 8bits, entre 20mV/div y 10V/div
- Modo de disparo: flanco, pulso, lógico, etc. + Externo
- Sonda: x10, 10pF, 10MΩ mediante BNC

Además de esto, también podemos grabar formas de onda y almacenarlas en archivos disponibles para ser tratados con Matlab o Excel. Su precio oficial es de 299,00€.

### *Osciprime*

Este proyecto es una plataforma hardware para la adquisición de datos a alta velocidad basada en Android, *Open Source*. La plataforma es capaz de conectarse a dispositivos Android a través de su puerto USB y convertir un Smartphone o Tablet en un auténtico osciloscopio.

El proyecto se inició en la Universidad de Ciencias Aplicadas del Noroeste de Suiza en 2010, donde fue galardonado con el premio a la mejor tesis de licenciatura de la Facultad Técnica. Desde entonces ha sido desarrollado y mantenido por *Nexus-Computing Suiza* donde sus creadores trabajan como ingenieros. A pesar de que el proyecto se define como *Open Source* y tanto el código software como el esquema hardware está disponible para cualquier persona, también es posible comprar tanto la placa *OsciPrime* como la app Android directamente.

El hardware consta de dos partes: el *front-end* analógico donde se captura una señal y se adapta para los conectores A/D, y un *front-end* digital donde los datos son cuantificados y preparados para la transmisión por USB. Los componentes clave son un *CPLD Xilinx Coolrunner* (Dispositivo Lógico Programable Complejo) y un microcontrolador *Cypress FX2*.

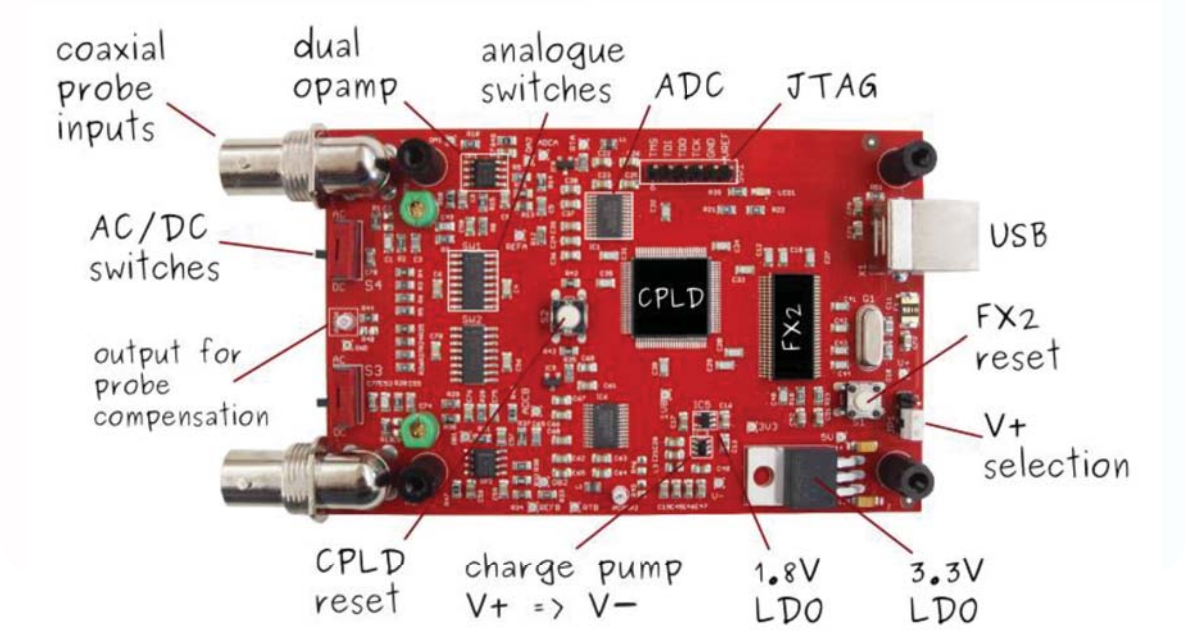



***Figura 5. Placa controladora del Osciprime***

Como hemos dicho, la forma en la que el hardware se conecta a nuestros terminales Android es mediante USB, haciendo uso dela API de Android para el USB Host, con lo que está limitado el uso a los dispositivos compatibles con este estándar.

De momento lo han testeado con un *Acer A500*, un *Galaxy Nexus* y con la *Nexus7*, utilizando un convertidor OTG, todos ellos incluso sin la necesidad de tener el acceso "*root*" activado. Aunque los propios desarrolladores advierten que aquellos Smartphones o Tablets que hagan uso del USB OTG deberán tener en cuenta que no dispondrán de una fuente externa, por lo que al utilizar el osciloscopio gastarán la batería rápidamente.

La aplicación para Android puede comprarse y se encuentra disponible en *Google Play*, aunque también podemos acceder al código fuente como dijimos anteriormente.

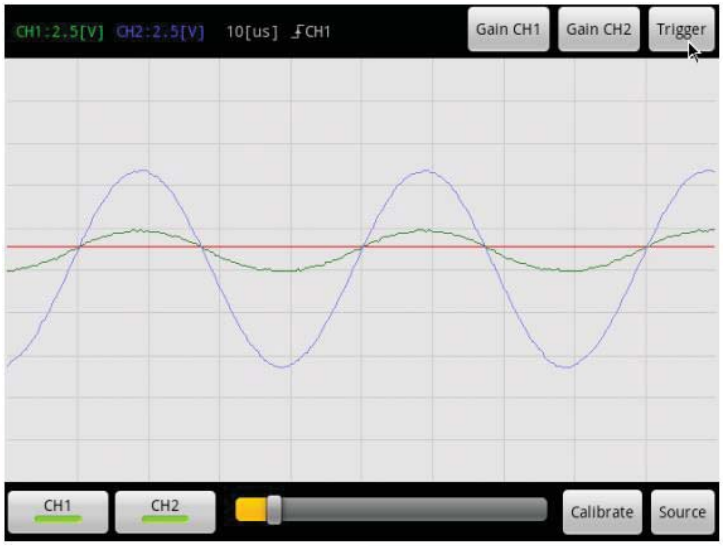



***Figura 6. Interfaz de usuario de la app del Osciprime***

Por otra parte, en su informe técnico disponible en su web, podemos entender mejor su funcionamiento (hardware y software), así como encontrar entre otras cosas sus especificaciones como osciloscopio, que es el tema que nos ocupa:

- Ancho de banda de 3,3MHz a 8MHz (dependiendo de la ganancia)
- Tasa de muestreo de hasta 48MHz
- Dos canales
- Acoplo de entrada DC, AC y GND
- Resolución vertical de 8bits, con $V_{in}$ máximo de ±16V
- Resolución horizontal de 5µs/div a 1ms/div
- Modo de disparo: flanco, CH1/CH2
- Medidas: Voltaje, Frecuencia y Tiempo
- Sonda: x10, mediante BNC
- Alimentación y transmisión de datos por USB
- Consumo total aproximado de 1W (880mW)

Su precio va desde los 269,00€, aunque actualmente se encuentra fuera de stock.

## Proyectos DIY de osciloscopios digitales basados en Android

A continuación, veremos algunos osciloscopios digitales DIY (*"Do It Yourself"*, Hágalo usted mismo) desarrollados por diferentes usuarios, analizando diversos puntos, como sus funcionalidades y características, su forma de adquirir y transmitir los datos y su interfaz de usuario, de manera que podamos establecer cuáles son sus puntos fuertes y débiles en comparación con el resto.

### Proyectos basados en la entrada de audio

Existen numerosos proyectos en Internet basados en la captación de señales para su posterior visualización a través de la entrada de audio de los terminales Android. Sin embargo, la mayoría utilizan una aplicación que analiza el espectro de audio y simplemente te dan una orientación de la frecuencia a la que se está reproduciendo la señal de entrada, por lo que resultan extremadamente limitados en cuanto a funcionalidad.

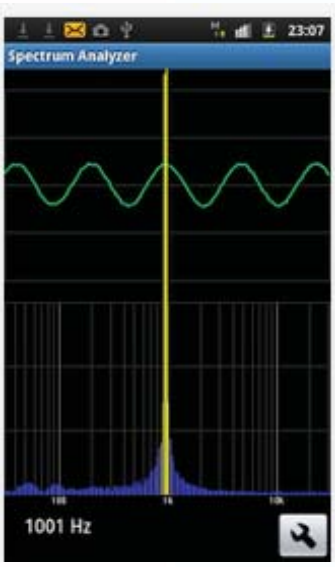



***Figura 7. Salida en app para analizar espectros de audio captado por el micrófono***

Nosotros vamos a centrarnos en proyectos más elaborados como pueden ser los siguientes.

#### *iPod Oszi*

Este es un proyecto creado por un usuario alemán, que se basa en la construcción de una sonda para medir y adaptar señales a la entrada de audio de los dispositivos móviles Android e iOS, para su posterior análisis en el mismo.

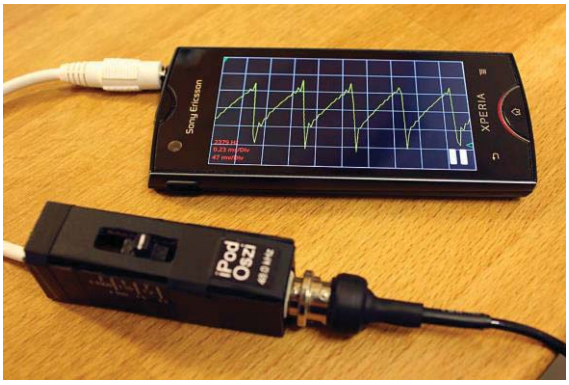
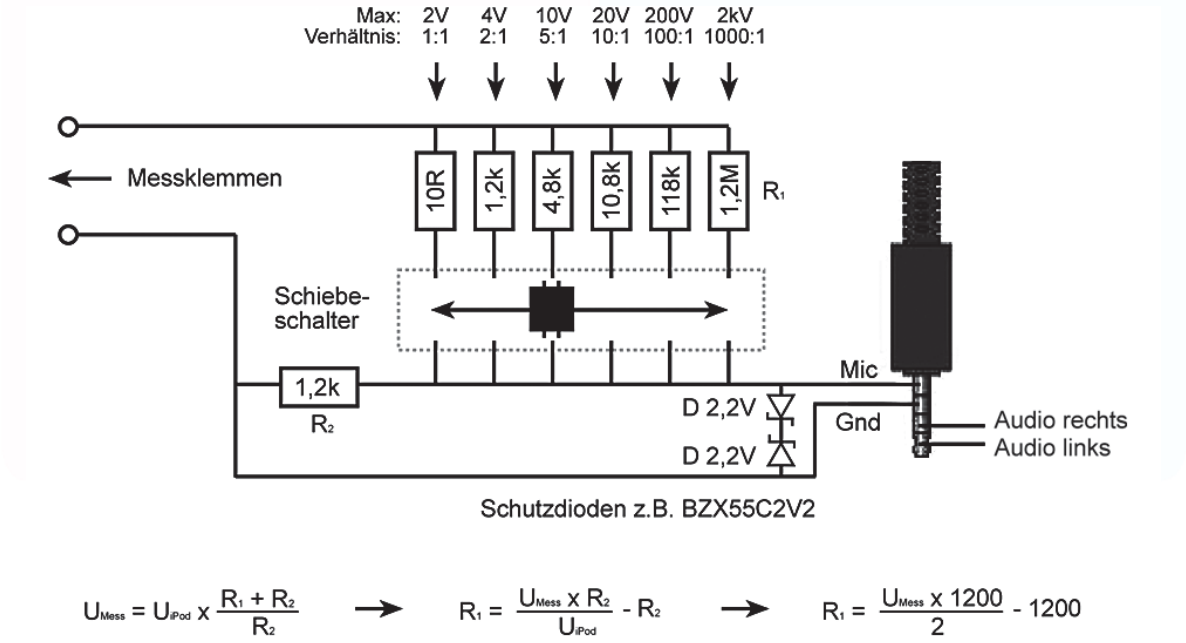



*Figura 8. Sonda iPod Oszi conectada a Smartphone*

Para poder utilizar esto como un osciloscopio se utiliza la entrada de micrófono, lo cual permite una medición de hasta unos 44,1 – 48KHz.
Sin embargo, esta entrada de micrófono solo puede manejar tensiones de hasta 2,1V. Por ello, necesitamos de esta sonda, para adaptar las tensiones y poder medir tensiones más altas. El esquema que utiliza es muy simple y se basa en un divisor de tensión, donde una de las resistencias es fija (1200Ω) y la otra es variable y seleccionable mediante un interruptor deslizante, que nos muestra las diferentes relaciones de voltaje de entrada/salida.




$$U_{Mess} = U_{iPod} \times \frac{R_1 + R_2}{R_2} \rightarrow R_1 = \frac{U_{Mess} \times R_2}{U_{iPod}} - R_2 \rightarrow R_1 = \frac{U_{Mess} \times 1200}{2} - 1200$$

*Figura 9. Diagrama eléctrico del circuito de la sonda*

El motivo, según explica este usuario, de colocar una resistencia fija de este valor, es que la mayoría de los dispositivos Android e iOS necesitan de una impedancia de entrada de aproximadamente 1KΩ para detectar la entrada de micrófono.

Como está pensado para que podamos medir tensiones más altas, no se debe descuidar la seguridad frente a sobrevoltajes, por lo que ha instalado también dos diodos Zener justo a la entrada de la clavija de entrada al terminal.

Como vemos, las ventajas de este proyecto son:

- Fácil desarrollo y producción
- Proyecto económico
- Gran rango de tensiones disponible
- Versatilidad en gran número de terminales móviles

No obstante, también presenta grandes desventajas o limitaciones:

- El terminal móvil y la sonda no están aislados eléctricamente
- El rango de frecuencias a medir solo llega hasta unas decenas de KHz

Por último comentar que la aplicación con la que el usuario realiza las medidas es *"Oscilloscope Pro"*, de *NFX Development* y que tiene un precio en *Google Play* de 5,99€. Ya que varias de las alternativas estudiadas utilizan esta aplicación, comentaremos su interfaz de usuario más adelante.

### *OscilloPhone: Oscilloscope / Signal Generator*

En este caso, el usuario que ha desarrollado el proyecto ha creado una estación portátil que hace las veces de generador de funciones y osciloscopio todo en uno.

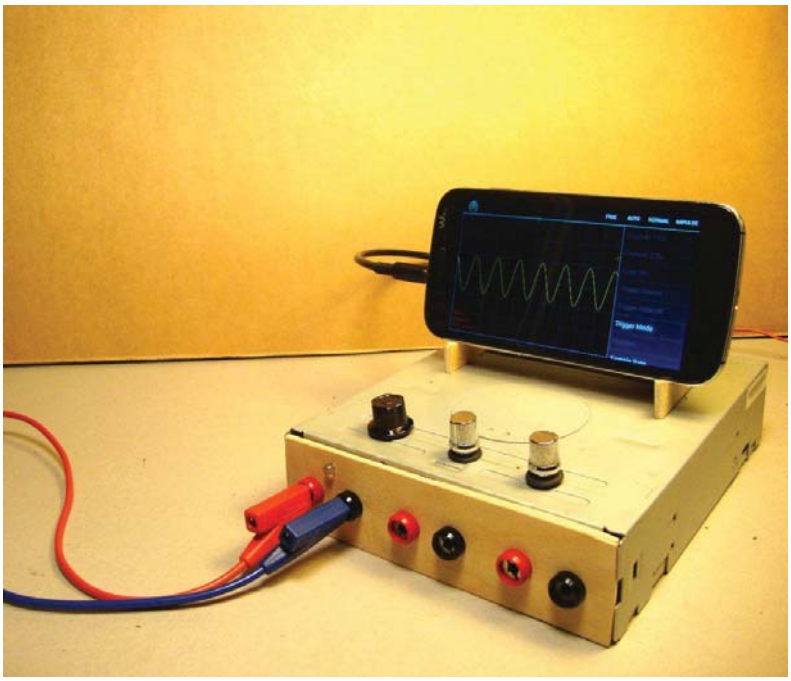


***Figura 10. Estación portátil OscilloPhone conectada y midiendo***

Se conecta con el terminal Android mediante el jack de audio e incluye:

- Una entrada de osciloscopio
  - Desde 150Hz a 15KHz (Frecuencias mayores tendrán peor calidad de señal)
  - Para entradas de hasta ±50V (Entradas mayores a través de circuito auxiliar)
- Dos salidas del generador de funciones
  - Una para señales cuadradas, triangulares y sinusoidales hasta 15KHz
  - Otra para las mismas señales, pero de mayor potencia (corrientes hasta 2A)

En este caso el usuario, pone en nuestro conocimiento dos aplicaciones para Android para utilizar el generador de funciones (una de pago y otra gratuita) y dos más para la opción del osciloscopio, de nuevo una de ellas la de *"Oscilloscope Pro"*.

En cuanto al circuito electrónico que utiliza para adecuar las señales de entrada y salida del dispositivo Android, solo nos centraremos en la parte del osciloscopio, que es la que nos ocupa.

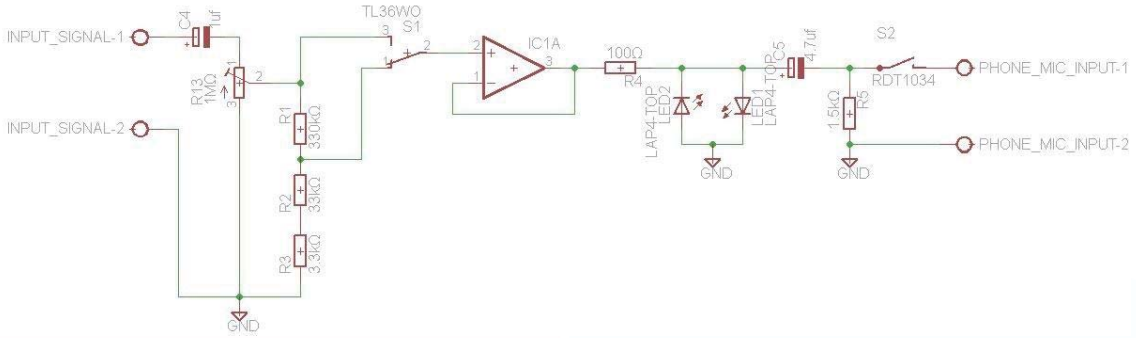



***Figura 11. Diagrama eléctrico del circuito para la sonda del osciloscopio***

En el circuito podemos diferenciar 4 etapas:

- Condensador de 1µF, potenciómetro y *array* de resistencias: se utilizan para reducir el voltaje de entrada hasta uno apto para la entrada de audio del terminal, así como bloquear la componente DC de la señal.
- Amplificador operacional: se usa como buffer para ajustar la impedancia del circuito.
- Resistencia de 100Ω y LEDs: Se utilizan para limitar el voltaje que alcance al terminal y avisarnos de si este es más alto de la cuenta para que utilicemos la reducción de la primera etapa (interruptor S1).
- Condensador de 4,7µF y resistencia de 1,5KΩ: son utilizados para aislar la entrada del micrófono y para que el terminal Android reconozca una fuente externa conectada.

Al igual que en el proyecto anterior, observamos que al utilizar la entrada de audio del terminal móvil, solo podemos procesar señales de hasta unas decenas de KHz, ya que a partir de la mitad del ancho de banda total del procesador, las señales serán más débiles y de peor calidad. Otra de las limitaciones que tiene el proyecto es que no se pueden usar los dos aparatos a la vez, excepto en aquellos dispositivos que permitan la multitarea.
Como sugerencias de mejora se destaca el posible uso de optoacopladores para aislar completamente de manera eléctrica el circuito del terminal.

### *Oscilloscope Pro*

Esta alternativa desarrollada por *NFX Development* es el proyecto del que surgió la aplicación para dispositivos Android mencionada en proyectos anteriores y que toma el mismo nombre, y es utilizada como herramienta para los ingenieros electrónicos o de audio, de manera que se puedan analizar y monitorizar señales electrónicas o de audio.

Originalmente el proyecto estaba enfocado a la conexión mediante la entrada de micrófono de los dispositivos Android, aunque más recientemente se ha dado soporte a la conexión mediante USB, la cual comentaremos más adelante.

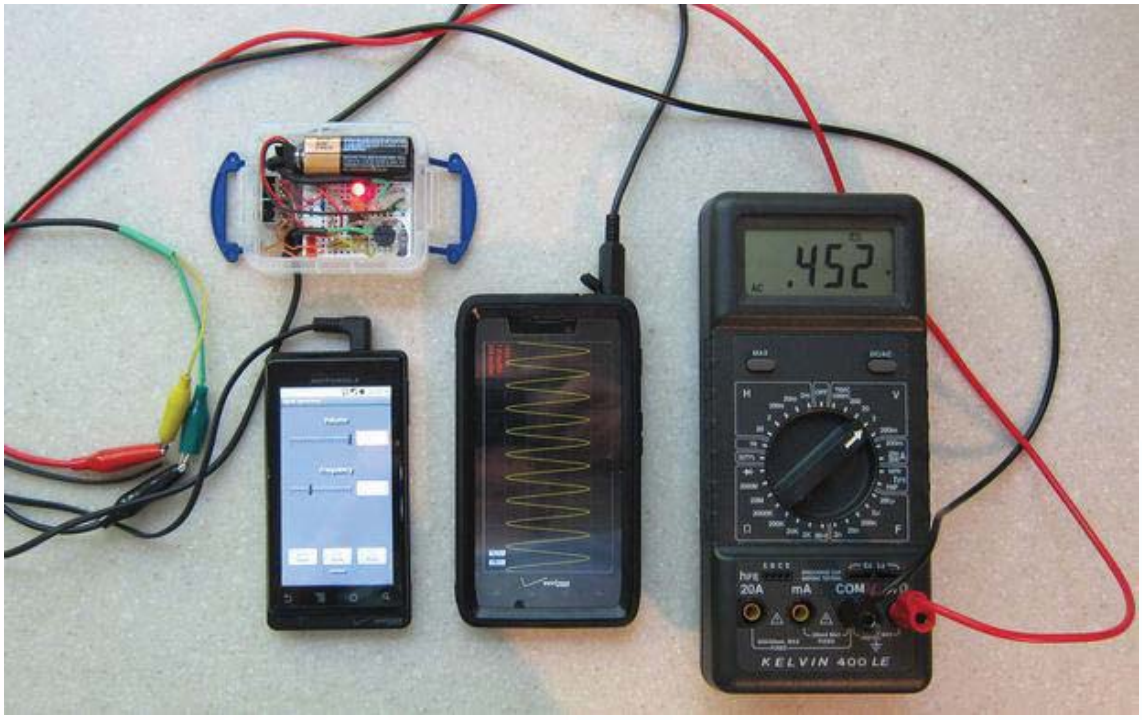



***Figura 12. Oscilloscope Pro midiendo a través de la entrada de audio***

Básicamente se dedicó el desarrollo a la fabricación de un preamplificador que adaptase las señales de interés, a la entrada de audio de los Smartphones mediante el jack de 3.5mm. Y es que, aunque estemos analizando el mismo concepto de soluciones anteriores, es de este proyecto del que toman idea el resto.
El objetivo del proyecto fue realizar un osciloscopio para móviles más versátil, útil y altamente resistente a transferencias de voltaje dañinas para nuestros terminales.

En cuanto a la fabricación del preamplificador, primero se partió de un circuito simple al que se le fueron añadiendo diversas mejoras:

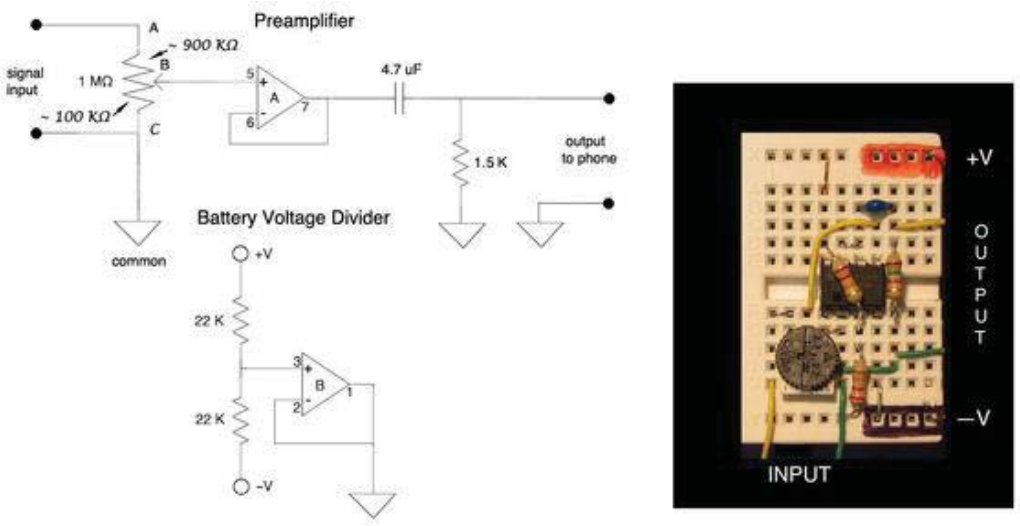



*Figura 13. Diagrama del circuito básico*

Este primer circuito es el núcleo principal y consta de dos operacionales y un potenciómetro, además de algunas resistencias y condensadores. El funcionamiento principal es:

- El operacional B es un divisor de tensión que da a su salida el voltaje mitad de la entrada invertido.
- El potenciómetro fija la impedancia de entrada de manera que no le llegue un sobrevoltaje al operacional A, el cual sirve de buffer adaptando la impedancia en su salida.
- Por último el filtro formado por el condensador y la resistencia, aíslan la entrada del micrófono de la componente DC del preamplificador y sirve para que el dispositivo reconozca la fuente de entrada externa.

Al analizarlo, vemos que tiene sus limitaciones, como que solo funciona con voltajes de unos ±10V a la entrada, el jack del dispositivo está expuesto a transitorios de voltajes altos.

Para mejorar estos aspectos, se llevó a cabo una ampliación del circuito con algunos elementos a mayores.

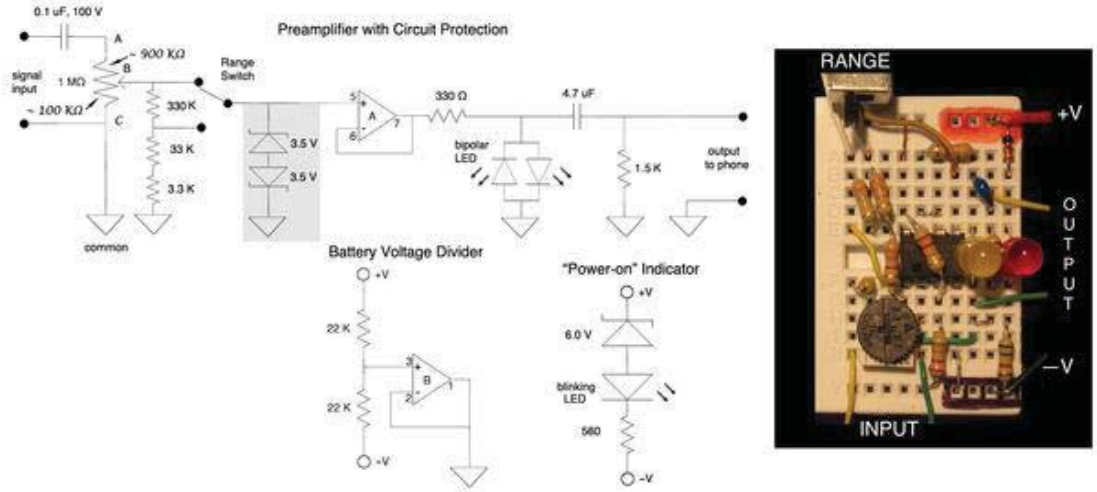



***Figura 14. Diagrama de la ampliación del circuito (circuito final)***

En este circuito se incluyeron:

- Una serie de resistencias controladas por un interruptor de dos posiciones, con las cuales podemos obtener un ratio de atenuación de 10 a 1 adicional, con lo cual aumentamos el rango de valores a la entrada en un factor 10. Y un condensador a la entrada para eliminar la componente continua de las señales de entrada.
- Un par de diodos Zener (parte gris en el diagrama) que ofrecen un nivel adicional de protección, pero que aumentan el ruido del circuito.
- La resistencia de 330Ω y los diodos LED para limitar el voltaje a la entrada del micrófono y advertir voltajes mayores.
- Un indicador de batería para monitorizar el estado de la misma y saber cuándo está baja, aunque esto último no nos importa mucho para nuestro estudio.

En cuanto a la aplicación, tiene tres modos diferentes que son accesibles presionando el botón de menú de los dispositivos:

- Auto: Muestra continuamente la señal de entrada, aunque también puede disparar señales que excedan el umbral fijado.
- *Trigger*: Designado a monitorear señales continuas, como senos o cosenos.
- *Impulse*: Tiene un rango de tiempo mayor que el modo anterior. También posee un indicador que puede fijarse para disparar solo una vez y detener la monitorización. En este modo se puede arrastrar la línea de tiempo y ver los datos después del disparo.

Las lecturas que nos da son de voltaje y frecuencia, aunque también nos permite colocar cursores para poder medir tiempos en las señales o amplitudes.

Una vez vistas diferentes soluciones basadas en la entrada de audio, podemos intentar resumir sus ventajas y limitaciones.

- Ventajas
  - No necesitamos de un procesador externo
  - Valido para cualquier terminal con conexión jack de 3,5mm (la gran mayoría)
  - Componentes para el circuito de adaptación muy económicos
- Limitaciones
  - Solo disponemos de un canal
  - Necesidad de circuitos de protección y adaptación para rangos de voltaje elevados
  - Terminal no aislado eléctricamente del resto del circuito
  - Solo valido para rango de frecuencias pequeño
  - Solo para acoplo AC

## Proyectos basados en USB

### *Oscilloscope Pro*

Para empezar, vamos a analizar el mismo proyecto anterior que, como habíamos dicho, más recientemente ha incluido un nuevo soporte para la conexión por USB entre el circuito y el terminal Android y que según comentan los desarrolladores es una mejor opción de entrada de datos.

La solución se basa en el uso del modo *USB Host* en los terminales y un hardware específico para la conexión USB proporcionado por *Gabotronics* llamado *XMEGA Xprotolab*, un dispositivo que para sus pequeñas dimensiones (2,5*4cm) cuenta con 2 entradas analógicas y 8 digitales (para su uso como analizador lógico), y numerosas funciones.

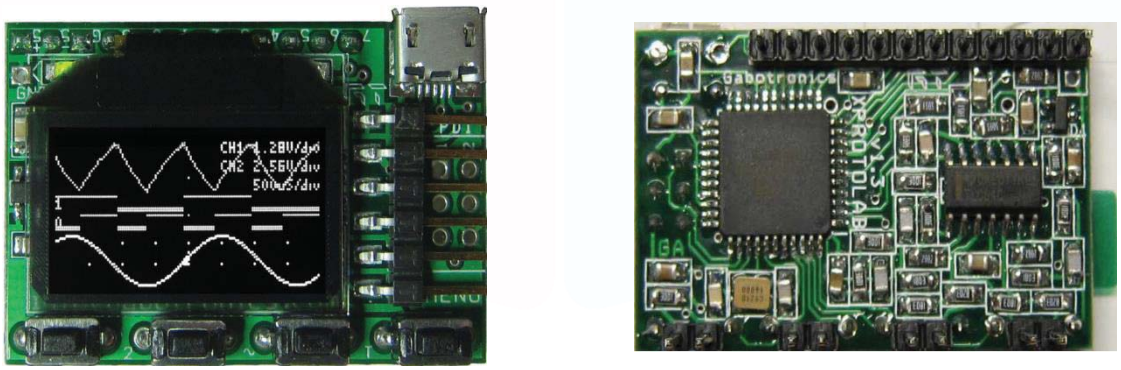



***Figura 15. Placa de circuito impreso de la tarjeta XMEGA Xprotolab***

Entre sus principales características (nos centramos en su uso como osciloscopio) y funciones se encuentran:

- Características
  - 2 canales
  - Máxima tasa de muestreo de 2MS/s
  - Ancho de banda de hasta 200kHz

- Resolución de 8 bits
- Impedancia de entrada de 1MΩ
- Rango de voltaje de entrada de -14V a 20V

- Funciones
  - Modos de disparo
    - Normal
    - Simple (con umbral)
    - Auto
    - Flanco
  - Medición de amplitudes (media y pico a pico) y frecuencias
  - Modo XY
    - Figuras de *Lissajous*
    - Diferencias de fase
    - Curvas V/I
  - Cálculos de FFT

En cuanto a la aplicación, en este caso tiene cuatro modos diferentes que están resaltados en color azul:

- *Free*: Sin disparos
- Auto: Igual que en el caso anterior con el jack
- *Trigger*: También como en el caso anterior
- *Impulse*: De nuevo como en el caso anterior

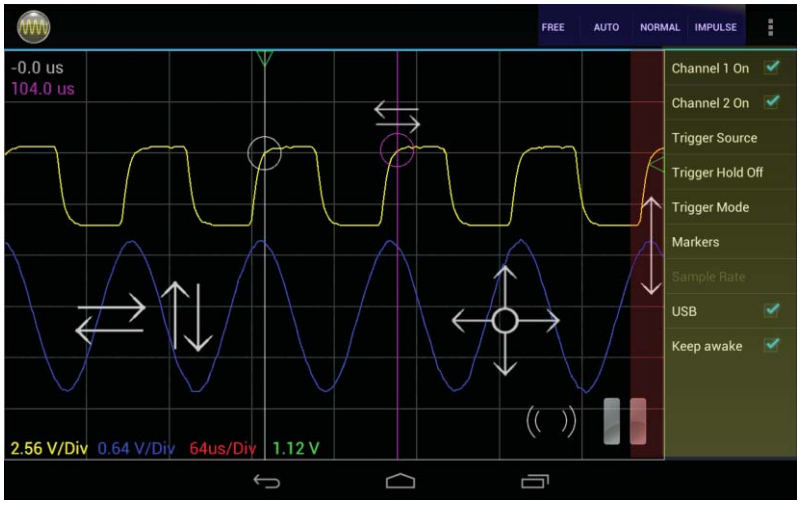



***Figura 16. Interfaz de usuario de la app Oscilloscope Pro conectada por USB***

La diferencia es que como ahora podemos representar dos señales, además de las medidas de amplitud y frecuencia, también podemos realizar medidas entre señales: tiempos de subida, desfases, etc.

Con este tipo de conexión para el dispositivo Android, encontramos algunas mejoras y algunos inconvenientes con respecto a la opción de la entrada de audio:

- Ventajas
  - Aumentamos el rango de frecuencias que podemos medir
  - Podemos muestrear más rápido
  - Aumentamos el número de canales → Operaciones entre señales
- Inconvenientes
  - Necesitamos procesador externo → Componentes más caros
  - No todos los terminales son compatibles
  - Seguimos necesitando circuitos de protección contra sobrevoltajes

Por último destacar que desde la página de descarga de la aplicación, nos indican que solo ha sido testeada en algunos modelos (*Samsung Galaxy Nexus, Asus Transformer, Asus Nexus 7, Samsung Galaxy S3*) y que es probable que necesitemos un cable USB OTG para conectarlo.

## Proyectos basados en Bluetooth

Finalmente veremos alguna alternativa que utiliza la conexión mediante Bluetooth como medio de transmisión de la información.

### *OscBox*

Este es un proyecto desarrollado por *Oltox Ltd. Co.*, el cual consta de dos partes, un hardware (el *OscBox-1600*) y la aplicación software *AR-Oscilloscope*, que combinadas nos proporcionan un osciloscopio Android por Bluetooth.
Este dispositivo puede ser utilizado además en otros sistemas como Windows, Linux o Mac si nuestro ordenador tiene conexión inalámbrica Bluetooth.

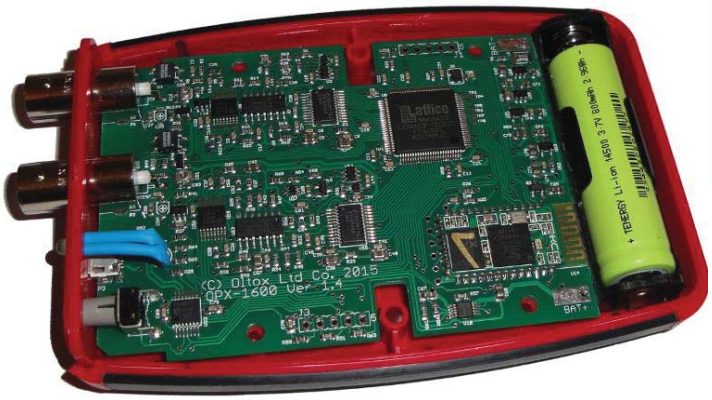



***Figura 17. Hardware del OscBox-1600***

El dispositivo es lo suficientemente pequeño para poder introducirlo dentro de cualquier sistema móvil. Esto abre nuevas posibilidades de medición para cualquier persona que trabaje en campos como robótica o drones. Además, la API de Java está disponible de manera que cualquiera puede construir su propio software e implementar cualquier análisis de datos que necesite.

Las medidas que se pueden realizar con este dispositivo son:

- Tensión continua/alterna
- Corriente continua/alterna
- Tiempos y retardos
- Fases y diferencias de fase
- Frecuencias

Básicamente la captura de señales se realiza en el hardware que hemos visto, donde se procesan y se realizan los cálculos. A continuación, los resultados son enviados por Bluetooth al terminal de destino donde son representados, según las necesidades del usuario.
Esto lo hace extremadamente portable, intuitivo y fácil de usar. Seguidamente podemos ver la interfaz de usuario, donde vienen representadas las diferentes medidas tomadas.

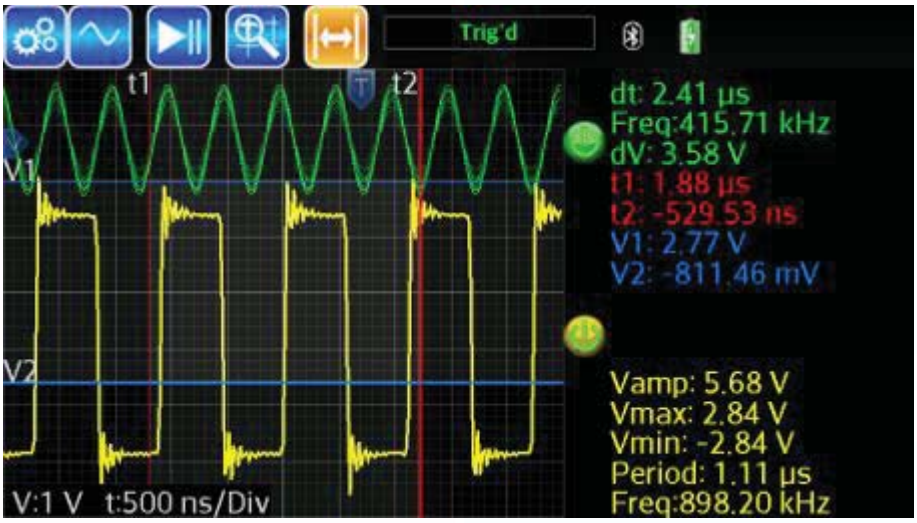



*Figura 18. Interfaz de usuario de la app AR-Oscilloscope*

*Android Bluetooth Oscilloscope*

Esta última solución también está basada en Bluetooth como medio de transmisión. La placa controladora consta de un *dsPIC33FJ16GS504* para la conversión analógica a digital de los dos canales que tiene disponibles, así como su posterior procesado. Después los datos procesados son transmitidos al terminal Android a través de un módulo SPP Bluetooth *LMX9838*.

***Figura 19. Placa controladora del osciloscopio con móodulo Bluetooth***

En este caso el rango de voltajes de entrada depende de la configuración del preamplificador externo, pero por defecto tan solo tiene un rango de ±8V.
En cuanto a la resolución que muestra la aplicación tenemos:

- Vertical: 10mV, 20mV, 50mV, 100mV, 200mV, 500mV, 1V y 2V
- Horizontal: 5µs – 50ms

Como vemos en la siguiente imagen la interfaz es muy sencilla, ya que solo nos permite variar las escalas utilizadas, la posición vertical de las señales y controlar solo una señal cada vez. Además se puede pausar la captura para ver la instantánea de las señales.

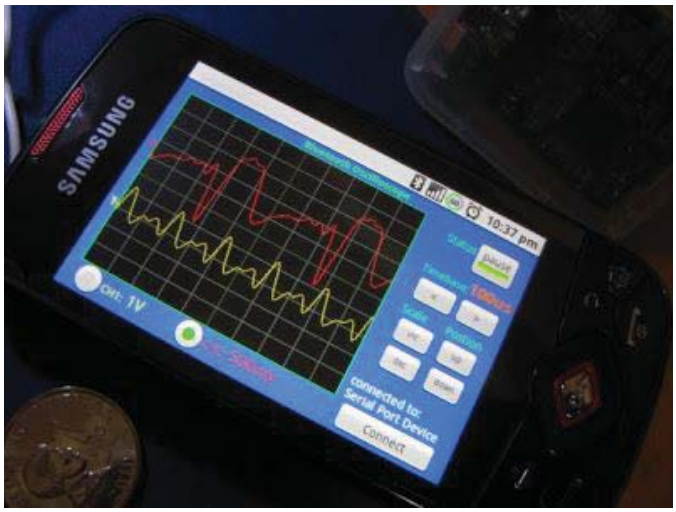



***Figura 20. Interfaz de usuario del osciloscopio***

A la vista de estas soluciones Bluetooth podemos sacar de nuevo nuestras conclusiones acerca de las diferencias que presentan con el resto de propuestas que utilizan un medio de transmisión distinto.

- Ventajas
  - El terminal Android y el circuito están aislados eléctricamente → No hay problemas de sobrevoltajes
  - Podemos representar más de una señal (Tantas como podamos capturar con nuestro circuito)
  - Muchos terminales son compatibles con Bluetooth
- Limitaciones
  - Necesitamos de un circuito de procesamiento externo → Encarece el precio total del proyecto
  - Necesitamos de un transmisor inalámbrico y por lo tanto adaptar los resultados para la transmisión con el mismo

## ¿Cuáles son las funciones más relevantes a incluir?

Una vez estudiadas las diferentes alternativas existentes, podemos hacer una lista de las funciones más importantes que utilizan estos dispositivos, en cuanto a medidas y caracterizaciones.

| | | | |
|---|---|---|---|
| **Sistema Vertical** | **Controles** | Acoplo | DC |
| | | | AC |
| | | | GND |
| | | Volts/Div. | |
| | **Medidas** | Tensiones | Pico |
| | | | Media |
| | | Cursores | Δ Volts |
| **Sistema Horizontal** | **Controles** | Tiempo/Div. | |
| | **Medidas** | Frecuencia | |
| | | Periodo | |
| | | Tiempos | Subida |
| | | | Bajada |
| | | Cursores | Δ Tiempo |
| **Sistema de *Trigger*** | **Controles** | Modo | Flanco |
| | | | Pulso |
| | | | Normal |
| | | | Auto |
| | | | Simple |
| | | | Externo |
| **Funciones** | Invertir CH | | |
| | +, -, *, / | | |
| | FFT | | |
| | X/Y (2 canales) | | |
| | Run/Stop | | |
| | Decodificador de protocolos | | |
| | Medida de fases | | |

## Caracterización del hardware disponible

En este próximo apartado estudiaremos cuáles son las características del hardware del que disponemos, para analizar cuál podría ser la opción más interesante que implementar.

### *Samsung Galaxy S4 mini GT-I9195*

En concreto el modelo del que disponemos para realizar las pruebas es un Smartphone de gama media de Samsung denominado *Galaxy S4 mini*, cuya aparición en el mercado data del año 2013. Se trata de una versión más compacta del modelo de gama superior *Samsung Galaxy S4*.

Sus principales características son las siguientes:

- Tamaño: 61,3*124,6*8,94mm
- Peso: 107g
- Micro-SIM
- Redes
  - GSM 850/900/1800/1900MHz
  - UMTS 850/900/1900/2100MHz
  - LTE 800/850/900/1800/2100/2600MHz
- Redes móviles
  - UMTS
  - EDGE clase 12
  - GPRS clase 12
  - HSPA+
  - LTE Cat3
- Sistema Operativo
  - Android 4.2.2 *Jelly Bean* de fabrica
  - Android 4.4.2 *KitKat* actualizado
- Soc
  - *Qualcomm Snapdragon 400 MSM8930AB*
- Memoria
  - Interna 8GB
  - Ampliable por microSD/microSDHC/microSDXC
- Pantalla
  - Super AMOLED
  - 4,3”
  - 16:9
  - Resolución 540*960px
  - 256 ppi
  - 24 bits de color
  - Capacitiva, multitáctil y anti arañazos (*Corning Gorilla Glass 2*)
- Formatos de video
  - MPEG4, H.264, MP43, VC-1, WMV 7/8, Sorenson Spark, H.263, VP8
- Formatos de audio
  - MP3, AMR-NB / WB, AAC / AAC+ / eAAC+, WMA, OGG, FLAC, WAV
- Sensores

- Proximidad
- Luz
- Acelerómetro
- Brújula
- Giróscopo

- Cámaras
  - Principal CMOS BSI de 8MP (3264*2448p) 30fps
  - Secundaria de 2MP (1920*1080p)
- Conectividad
  - Radio RDS
  - GPS / A-GPS / GLONASS
  - Wi-Fi 802.11a/b/g/n, Wi-Fi Hotspot, Wi-Fi Direct
  - Bluetooth 4.0
  - Micro USB 2.0 (Carga (5V, 1A) y Almacenamiento Masivo)
  - NFC
- Batería
  - 1900mAh Li-ión Extraíble

Como nosotros estamos estudiando las diferentes formas mediante las cuales conectarnos con el terminal para transmitir o capturar los datos, nos fijamos en tres puntos principales:

- USB: Vemos que la conexión de nuestro terminal en este caso es Micro USB 2.0, aunque nos especifican que solamente se puede usar para la carga del dispositivo o para el uso del mismo como medio de almacenamiento masivo. En ningún sitio aparece que sea compatible con OTG.
  Para reforzar esta información, podemos hacer uso de aplicaciones con las que comprobar si nuestro dispositivo es compatible con OTG, como *USB OTG Checker*, una app gratuita con la que averiguaremos si nuestro Smartphone tiene la API necesaria.

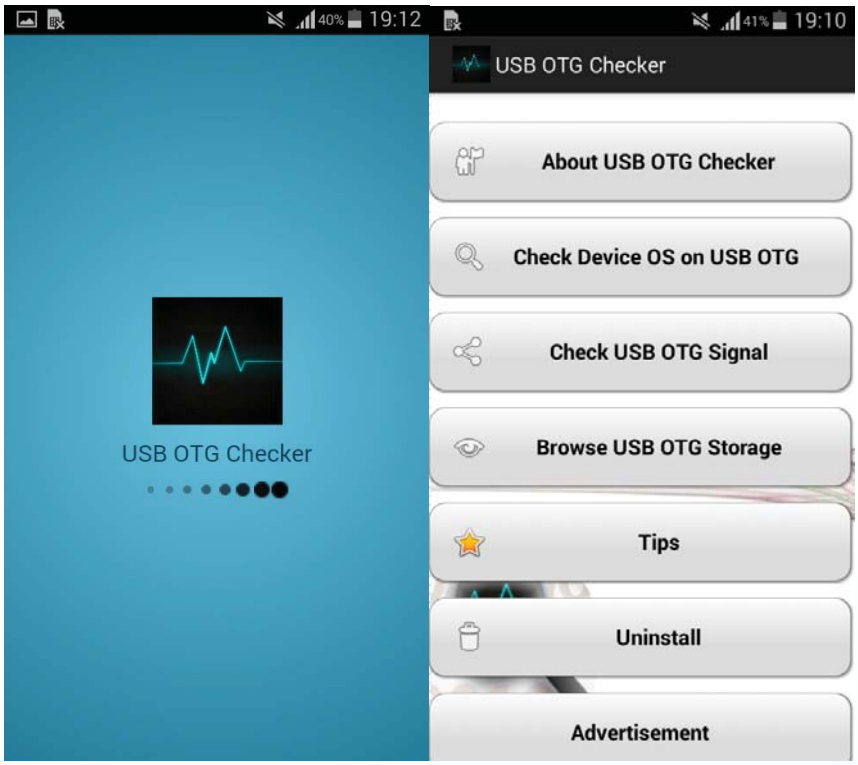



***Figura 21. Portada y menú principal de la app USB OTG Checker***

Una vez instalada la aplicación, procedemos a pulsar en *"Check Device OS on USB OTG"*.


***Figura 22. Resultado de la comprobación del OTG***

Y como vemos nos dice que nuestro dispositivo sí que tiene la API necesaria y que podemos proceder a probar si detecta la señal de un dispositivo conectado mediante USB OTG. Como no tenemos un cable compatible para poder probarlo, quedaremos a la espera de conocer la compatibilidad total del Smartphone por esta vía de comunicación.

- Bluetooth: En la sección referente a la conectividad Bluetooth solo nos especifican que la versión que ocupa este terminal es la 4.0, pero si indagamos un poco más, podemos conocer los perfiles y protocolos que se soportan en el dispositivo:
  - A2DP (Advanced Audio Distribution Profile)
  - AVRCP (Audio/Visual Remote Control Profile)
  - DIP (Device ID Profile)
  - HFP (Hands-Free Profile)
  - HID (Human Interface Profile)
  - HSP (Headset Profile)
  - LE (Low Energy)
  - MAP (Message Access Profile)
  - OPP (Object Push Profile)
  - PAN (Personal Area Networking Profile)
  - PBAP/PAB (Phone Book Access Profile)
  - SAP/SIM/rSAP (SIM Access Profile)

- Conector de Audio: Por ultimo nos fijamos en el conector de audio y vemos que el Smartphone cuenta con un conector jack de 3,5mm para auriculares desde el que podemos capturar las señales externas para ser procesadas. Por lo tanto, debemos fijarnos también en el procesador.
  En este caso cuenta con un *Qualcomm Snapdragon 400 MSM8930AB*, de 2 núcleos a una velocidad de reloj de 1,7GHz, que rinde bastante bien en multitarea.
  Tiene una arquitectura *Harvard* y su CPU principal es una *Qualcomm Krait 300*. Por otro lado, también posee una *GPU Qualcomm Adreno 305*, encargada de la creación acelerada de imágenes hacia la pantalla, procesando varios bloques en paralelo.
  Pero lo que nos interesa realmente el tratamiento de las señales analógicas como las de audio, video o banda ancha del móvil, y de eso se encarga el DSP especializado que monta este procesador, un *Qualcomm Hexagon QDSP6V4* a 500MHz. Este DSP se ocupa de procesar cada señal analógica convertida en digital, con una latencia muy baja y utilizando la menor potencia posible en el dispositivo. Es un DSP que puede ejecutar 3 hilos simultáneamente y que en nuestro caso también podría ser el encargado de realizar las FFTs.

Hasta lo visto ahora podríamos pensar que el ancho de banda de nuestra aplicación podría ser elevado, ya que la frecuencia de reloj del DSP es de 500MHz, sin embargo, aquí quien impone el cuello de botella es la entrada de micrófono, ya que está pensada para muestrear señales a unos 48KHz (aunque según varios artículos solo se garantiza una frecuencia máxima de muestro de 44.100Hz en todos los dispositivos Android), y como sabemos debemos cumplir el teorema de Nyquist, donde la frecuencia de muestreo debe ser al menos 2 veces la máxima frecuencia de la señal a

muestrear. A la vista de estos datos vemos que nuestra aplicación tendrá un ancho de banda, al igual que las que habíamos visto en soluciones anteriores, de unos pocos KHz (0 a 22.050Hz en sentido estricto).

Una vez analizado el hardware que tenemos a nuestra disposición, podemos intuir cuáles serán las características más comunes que se mantendrán en otros dispositivos similares. No obstante, sería útil el poder averiguar las características del dispositivo Android que estamos utilizando a través de la propia aplicación, y de esta manera, poder decidir ciertos aspectos de la interfaz de usuario como la resolución, o el número de controles en pantalla, o advertir al usuario de las limitaciones que tiene al usar ese dispositivo en concreto.

Para realizar esta tarea podemos ayudarnos de la clase Android *"Build"*, la cual nos permite obtener información acerca de la versión y características del terminal. A continuación, podemos ver algunas directivas utilizadas para poder obtener datos del sistema operativo y del dispositivo en concreto o para comprobar la resolución de pantalla.

```
System.getProperty("os.version"); // OS version
android.os.Build.VERSION.SDK      // API Level
android.os.Build.DEVICE           // Device
android.os.Build.MODEL            // Model
android.os.Build.PRODUCT          // Product

getWindow().getWindowManager().getDefaultDisplay().getWidth();
getWindow().getWindowManager().getDefaultDisplay().getHeight();
```

***Figura 23. Segmento de código para obtener los datos del dispositivo***

## Acceso a la toma de audio

El conector de audio, o jack de audio, como lo hemos venido llamando, se utiliza para conectar elementos que usan señales analógicas como micrófonos, auriculares y otros sistemas de señal analógica a dispositivos electrónicos.
Su principal aplicación es el audio (reproducir música) pero existe otro uso que no debemos descartar y es que también podemos utilizarlo para transmitir y recibir datos. Actualmente existen numerosos productos periféricos que utilizan el jack de audio para intercambiar información: glucómetros, controles remotos por infrarrojos, lectores de NFC, etc.

Cuando tratamos su uso en Smartphones o dispositivos móviles, donde los auriculares que conectamos también incluyen micrófono, se le denomina igualmente conector TRRS *("Tip-Ring-Ring-Sleeve"*, Punta-Anillo-Anillo-Cuerpo).
Aunque existen diferentes diámetros, la mayoría de dispositivos móviles utilizan el de 3,5mm y donde encontramos la principal diferencia es en el estándar que utiliza cada dispositivo, que tiene que ver con la posición del conexionado eléctrico. Existen dos estándares para la interfaz del jack de audio:

- OMTP: Se considera el estándar internacional

- CTIA: Es un estándar americano que utilizan entre otros los productos de Apple como iPhone o iPad

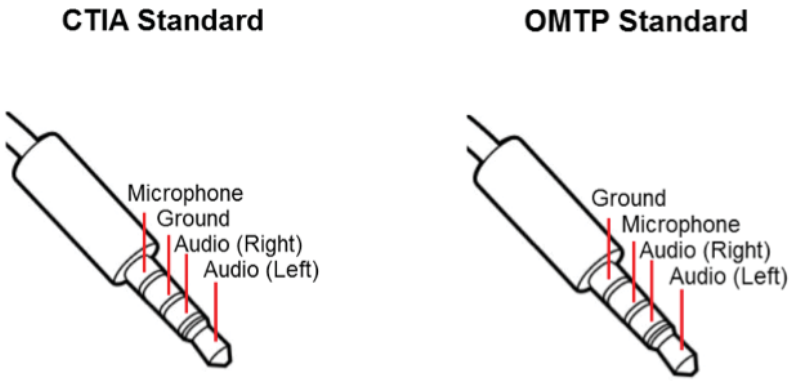



***Figura 24. Conexiones en los dos estándares del jack de audio***

En este apartado, nos vamos a centrar en explorar las opciones que tiene este conector para poder intercambiar información. La idea básica es hacer que nuestro dispositivo Android pueda transmitir información (actuando como reproductor de audio) o recibirla (actuando como grabador de audio), "transportando" esa información en ondas de audio.
Pero como sabemos, mientras que los datos son digitales, las ondas de audio son analógicas, por lo que necesitamos de algún tipo de modulación como la FSK.

## ¿Cómo transmitir datos?

Cuando queremos enviar datos a través de audio, el primer paso que debemos hacer es convertir la información digital a una señal analógica, es decir modular los datos.
Normalmente se usan portadoras sinusoidales para la señal analógica.

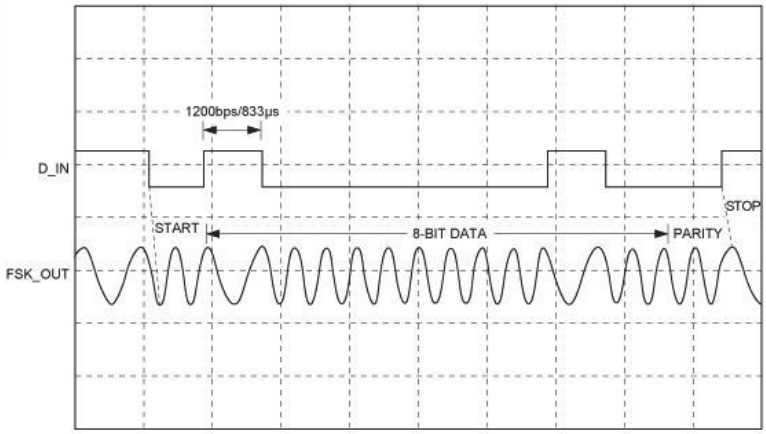



***Figura 25. Diagrama representativo para la transmisión de datos***

El siguiente paso, ya en el dispositivo Android, es llamar a la función de la API de audio "*audioTrack*" para reproducir el buffer que hayamos creado con la información. En el siguiente segmento de código podemos ver las sentencias necesarias para enviar datos desde el buffer usando la función citada anteriormente.

```
01 public void send(byte[] bytes_pkg) {
02         int bufsize = AudioTrack.getMinBufferSize(8000,
03                 AudioFormat.CHANNEL_OUT_MONO,
04                 AudioFormat.ENCODING_PCM_16BIT);
05         AudioTrack trackplayer = new AudioTrack(AudioManager.STREAM_MUSIC,
06                 8000, AudioFormat.CHANNEL_OUT_MONO,
07                 AudioFormat.ENCODING_PCM_16BIT, bufsize,
08 AudioTrack.MODE_STREAM);
09         trackplayer.play();
10         trackplayer.write(bytes_pkg, 0, bytes_pkg.length);
11     }
```

***Figura 26. Segmento de código para la transmisión de datos a través de la toma de audio***

Debemos comentar también que el rango de frecuencias disponibles para transmitir esa información dependerá del hardware que estemos utilizando.

## ¿Cómo recibir datos?

En este caso, para actuar como receptor necesitamos "traducir" la señal analógica recibida a valores digitales. Para ello debemos demodular la señal y decodificar los datos según el protocolo que estemos utilizando.

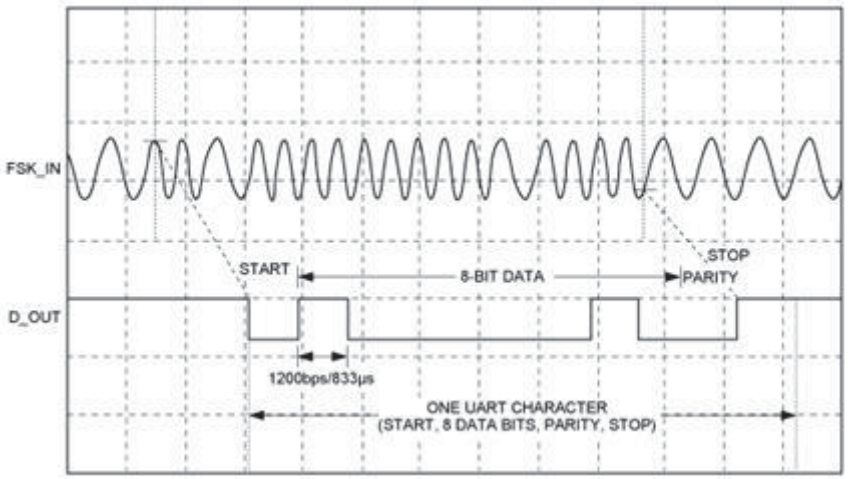



***Figura 27. Diagrama representativo para la transmisión de datos***

Cuando nuestro terminal recibe la señal analógica debe muestrearla para convertirla a digital. Como dijimos en apartados anteriores, el número total de veces que la señal es muestreada en 1s se define con la frecuencia de muestreo y para los dispositivos Android la máxima garantizada es de unos 44KHz. Como debemos cumplir el teorema de Nyquist la máxima frecuencia de la señal recibida ha de ser de la mitad, es decir de unos 22KHz.

En los dispositivos Android podemos usar la función "*audioRecord*" de la API de audio para grabar la señal de entrada. Con el siguiente segmento de código podemos muestrear las señales procedentes de la entrada del micrófono y grabar los valores en el buffer.

```
// construct AudioRecord to record audio from microphone with sample rate of 44100Hz
int minSize = AudioRecord.getMinBufferSize(sampleRate,AudioFormat.
                                           CHANNEL_CONFIGURATION_MONO,
                                           AudioFormat.ENCODING_PCM_16BIT);
AudioRecord audioInput = new AudioRecord(MediaRecorder.AudioSource.MIC, sampleRate,
                                           AudioFormat.CHANNEL_CONFIGURATION_MONO,
                                           AudioFormat.ENCODING_PCM_16BIT);
                                           minSize);
...
short[] buffer = new short[readSize];
audioInput.startRecording();
audioInput.read(buffer, index, readSize); // record data from mic into buffer
```

***Figura 28. Segmento de código para la recepción de datos a través de la toma de audio***

Por último, además de tener las muestras, con las que podemos representar las señales capturadas y operando con sus valores obtener sus parámetros característicos, también podemos utilizar un algoritmo para calcular la FFT (*Fast Fourier Transform*) y convertir nuestras señales al dominio de la frecuencia.

Como vemos, utilizar el conector de audio como medio de transmisión de información tiene sus ventajas:

- Cero transmisión de potencia
- Menos uso de batería comparado con Bluetooth o Wi-Fi
- Puede funcionar con cualquier dispositivo de audio
- Tan simple como conectarlo a un Smartphone o Tablet

## Planteamiento de la opción más adecuada

Finalmente, una vez analizadas las ventajas y limitaciones de cada alternativa tomamos una decisión sobre cuál sería la opción más adecuada a desarrollar en nuestro proyecto. Para ello nos basamos en una serie de puntos clave a tener en cuenta:

- La aplicación será desarrollada para un ámbito docente a nivel de Bachillerato y primeros cursos Universitarios, esto hace que debamos disponer de al menos 2 canales de entrada.
- El rango de frecuencias de las señales a analizar no deberá ser muy grande, pero si superar las decenas de KHz. Un rango adecuado estaría en torno a unos 50 – 100MHz.
- En el caso de necesitar una depuración a la hora de resolver problemas de malfuncionamiento del sistema, una alternativa con menor número de componentes será más fácil de analizar y resolver. Además, con circuitos más simples el diseño y fabricación del mismo será más sencillo.
- Cuanto más económico sea el sistema final, mayor número de ellos podremos conseguir.
- La idea de poder conseguir una mayor portabilidad hace que debamos utilizar baterías externas o conseguir la alimentación del dispositivo al que nos conectemos. En ambos casos, un menor número de elementos reducirá el consumo del circuito.

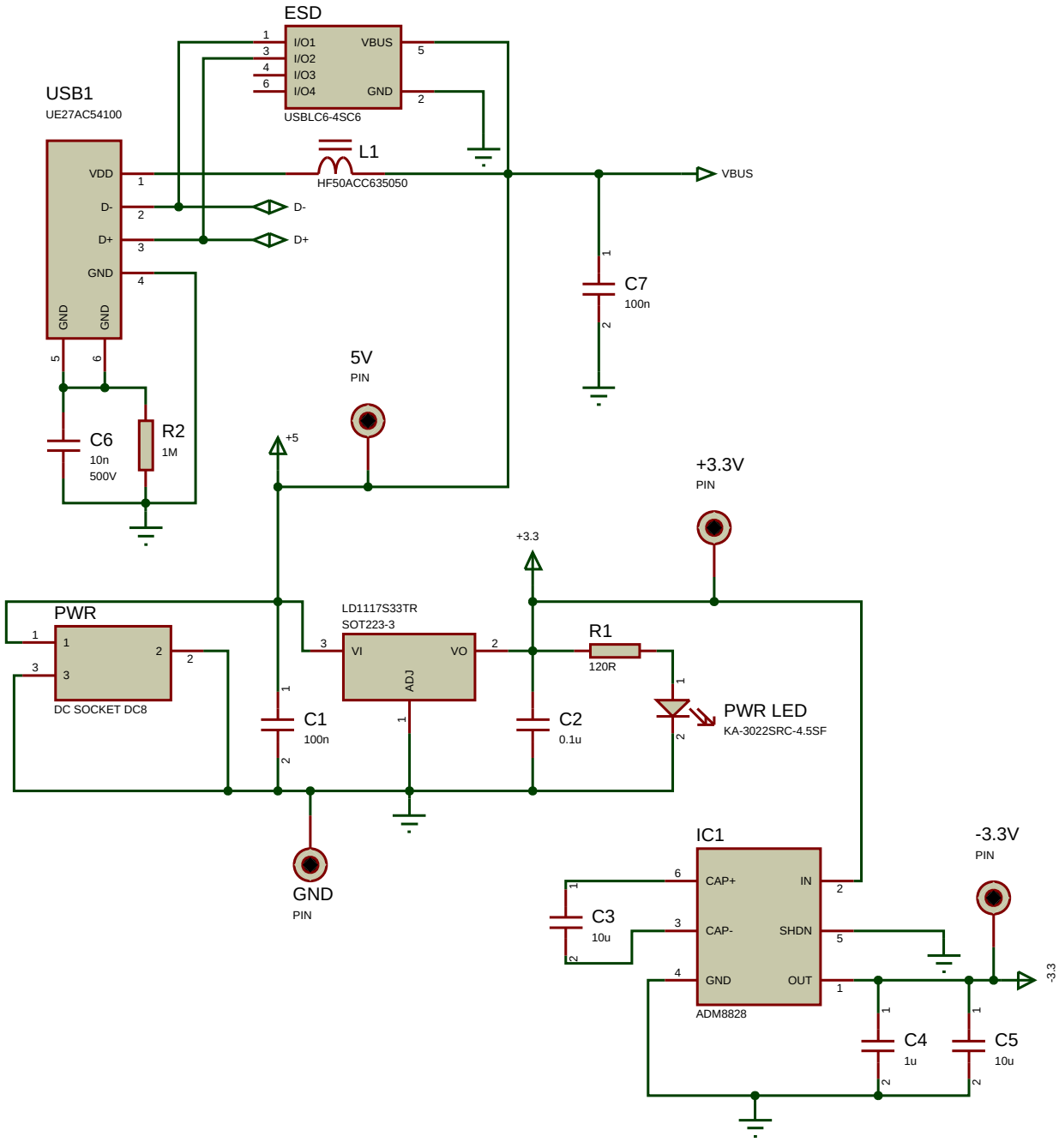
Por todas estas razones, la solución elegida es la realización de un osciloscopio digital conectado por USB (principalmente) al dispositivo Android, donde la captación y procesamiento de las señales a analizar se realizará mediante un DSP externo.

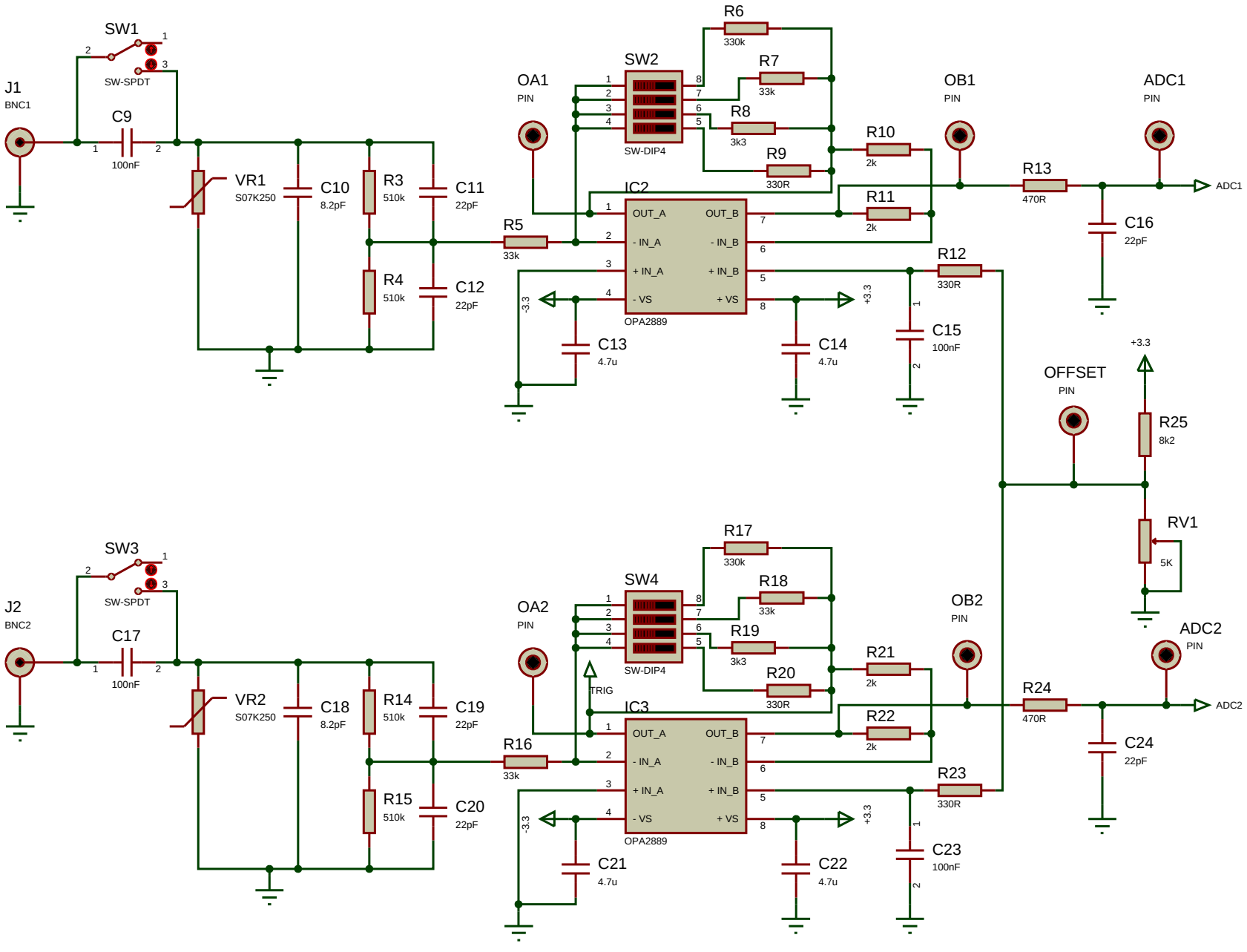
Las alternativas de transmisión de datos a través de la entrada de audio o mediante Bluetooth quedan descartadas debido a los inconvenientes relacionados con el número de canales, los rangos de frecuencias, o la mayor complejidad y coste del circuito.

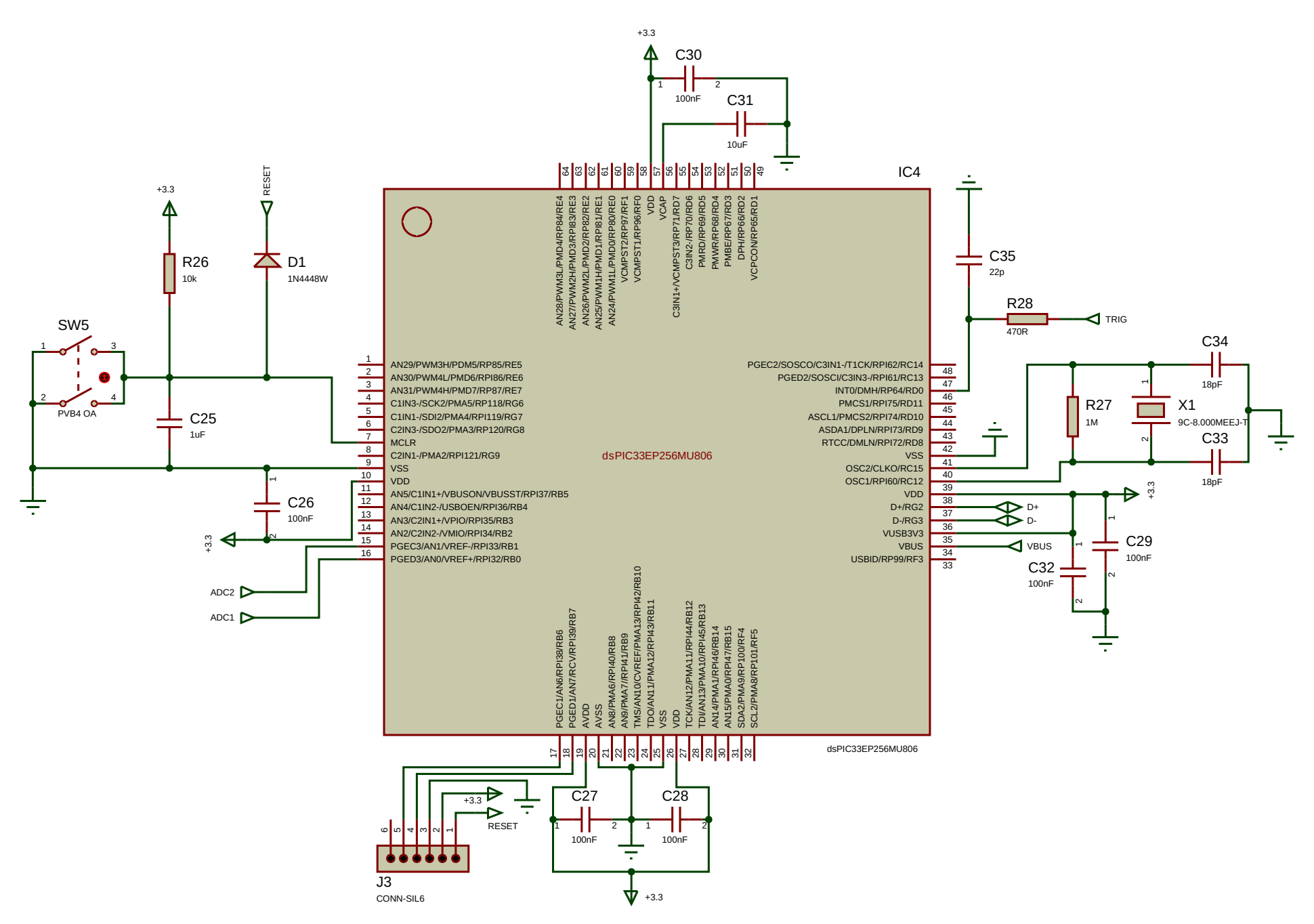
Aunque, debido a que cabe la posibilidad de que no todos los terminales soporten el estándar de USB escogido, se incluirá una solución alternativa basada en la conexión a través de la toma de audio, a pesar de las ventajas que ésta presenta.

# B. Diagramas eléctricos del sistema

Este anexo incluye los diagramas (esquemáticos) del circuito completo, realizados con la herramienta *Proteus ISIS*.


ESD
I/O1
I/O2
I/O3
I/O4
VBUS
GND
USBLC6-4SC6
USB1
UE27AC54100
VDD
D-
D+
GND
L1
HF50ACC635050
VBUS
C7
100n
5V
PIN
C6
10n
500V
R2
1M
+5
+3.3V
PIN
+3.3
PWR
DC SOCKET DC8
LD1117S33TR
SOT223-3
VI
VO
ADJ
R1
120R
PWR LED
KA-3022SRC-4.5SF
C1
100n
C2
0.1u
GND
PIN
IC1
CAP+
CAP-
GND
IN
SHDN
OUT
ADM8828
C3
10u
-3.3V
PIN
C4
1u
C5
10u
-3.3


SW1
SW-SPDT
J1
BNC1
C9
100nF
VR1
S07K250
C10
8.2pF
R3
510k
R4
510k
C11
22pF
C12
22pF
R5
33k
OA1
PIN
SW2
SW-DIP4
R6
330k
R7
33k
R8
3k3
R9
330R
R10
2k
R11
2k
IC2
OUT_A
OUT_B
- IN_A
- IN_B
+ IN_A
+ IN_B
- VS
+ VS
OPA2889
-3.3
+3.3
C13
4.7u
C14
4.7u
R12
330R
C15
100nF
OB1
PIN
R13
470R
ADC1
PIN
ADC1
C16
22pF
+3.3
OFFSET
PIN
R25
8k2
RV1
5K
SW3
SW-SPDT
J2
BNC2
C17
100nF
VR2
S07K250
C18
8.2pF
R14
510k
R15
510k
C19
22pF
C20
22pF
R16
33k
OA2
PIN
SW4
SW-DIP4
R17
330k
R18
33k
R19
3k3
R20
330R
R21
2k
R22
2k
IC3
TRIG
OUT_A
OUT_B
- IN_A
- IN_B
+ IN_A
+ IN_B
- VS
+ VS
OPA2889
-3.3
+3.3
C21
4.7u
C22
4.7u
R23
330R
C23
100nF
OB2
PIN
R24
470R
ADC2
PIN
ADC2
C24
22pF


+3.3
C30
100nF
C31
10uF
IC4
C35
22p
R28
470R
TRIG
C34
18pF
R27
1M
X1
9C-8.000MEEJ-T
C33
18pF
+3.3
C29
100nF
C32
100nF
D+
D-
VBUS
R26
10k
D1
1N4448W
RESET
SW5
PVB4 OA
C25
1uF
C26
100nF
ADC2
ADC1
dsPIC33EP256MU806
AN29/PWM3H/PDM5/RP85/RE5
AN30/PWM4L/PMD6/RPI86/RE6
AN31/PWM4H/PMD7/RP87/RE7
C1IN3-/SCK2/PMA5/RPI118/RG6
C1IN1-/SDI2/PMA4/RPI119/RG7
C2IN3-/SDO2/PMA3/RP120/RG8
MCLR
C2IN1-/PMA2/RPI121/RG9
VSS
VDD
AN5/C1IN1+/VBUSON/VBUSST/RPI37/RB5
AN4/C1IN2-/USBOEN/RPI36/RB4
AN3/C2IN1+/VPIO/RPI35/RB3
AN2/C2IN2-/VMIO/RPI34/RB2
PGEC3/AN1/VREF-/RPI33/RB1
PGED3/AN0/VREF+/RPI32/RB0
PGEC2/SOSCO/C3IN1-/T1CK/RPI62/RC14
PGED2/SOSCI/C3IN3-/RPI61/RC13
INT0/DMH/RP64/RD0
PMCS1/RPI75/RD11
ASCL1/PMCS2/RPI74/RD10
ASDA1/DPLN/RPI73/RD9
RTCC/DMLN/RPI72/RD8
VSS
OSC2/CLKO/RC15
OSC1/RPI60/RC12
VDD
D+/RG2
D-/RG3
VUSB3V3
VBUS
USBID/RP99/RF3
C27
100nF
C28
100nF
J3
CONN-SIL6

## *C.* Lista de materiales (*Bill Of Materials)*

En este anexo se detalla la lista de componentes utilizados, así como su precio, la cantidad mínima de compra en el/los distribuidor/es correspondientes y su código dentro del catálogo de proveedores.

# Bill Of Materials For OscAndroid

**Design Title** : OscAndroid
**Author** : David Ortiz de Latierro Delgado
**Revision** : 1

**Total Parts In Design** : 85

## 28 Resistors

| Quantity: | References | Value | Cod. Farmell/Mouser/RS | Pedido minimo | Precio/ud. |
|---|---|---|---|---|---|
| 1 | R1 | 120R | 2447561 | 10 | 0.0094 |
| 2 | R2, R27 | 1M | 2447596 | 10 | 0.0102 |
| 4 | R3, R4, R14, R15 | 510k | 2074481 | 10 | 0.018 |
| 4 | R5, R7, R16, R18 | 33k | 2447639 | 10 | 0.0098 |
| 2 | R6, R17 | 330k | 2447637 | 10 | 0.0102 |
| 2 | R8, R19 | 3k3 | 9334360 | 10 | 0.0098 |
| 4 | R9, R12, R20, R23 | 330R | 2695037 | 10 | 0.0096 |
| 4 | R10, R11, R21, R22 | 2k | 2447622 | 10 | 0.0099 |
| 3 | R13, R24, R28 | 470R | 2447662 | 10 | 0.0094 |
| 1 | R25 | 8k2 | 2447732 | 10 | 0.0098 |
| 1 | R26 | 10k | 2447553 | 10 | 0.0098 |

## 34 Capacitors

| Quantity: | References | Value | Cod. Farmell/Mouser/RS | Pedido minimo | Precio/ud. |
|---|---|---|---|---|---|
| 12 | C1, C7, C9, C15, C17, C23, C26-C30, C32 | 100nF | 1759167 | 10 | 0.102 |
| 4 | C2, C3, C5, C31 | 10uF | 1759421 | 5 | 0.151 |
| 2 | C4, C25 | 1uF | 1759422 | 10 | 0.0549 |
| 1 | C6 | 10nF | 1284130 | 5 | 0.156 |
| 2 | C10, C18 | 8.2pF | 2896528 | 10 | 0.0584 |
| 7 | C11, C12, C16, C19, C20, C24, C35 | 22pF | 1759195 | 10 | 0.0571 |
| 4 | C13, C14, C21, C22 | 4.7uF | 1759478 | 10 | 0.0885 |
| 2 | C33, C34 | 18pF | 2320820 | 10 | 0.0549 |

## 1 Diodes

| Quantity: | References | Value | Cod. Farmell/Mouser/RS | Pedido minimo | Precio/ud. |
|---|---|---|---|---|---|
| 1 | D1 | 1N4448W | 1097176 | 5 | 0.109 |

## 22 Miscellaneous

| Quantity: | References | Value | Cod. Farmell/Mouser/RS | Pedido minimo | Precio/ud. |
|---|---|---|---|---|---|
| 1 | ESD | USBLC6-4SC6 | 710-82400274 | 1 | 0.90 |
| 1 | IC1 | ADM8828 | 1651323 | 1 | 2.14 |
| 2 | IC2, IC3 | OPA2890 | 709-3147 | 1 | 3.86 |
| 1 | IC4 | dsPIC33EP256MU806 | 1971850 | 1 | 6.21 |
| 1 | J1 | BNC1 | 2779837 | 1 | 1.72 |
| 1 | J2 | BNC2 | 2779837 | 1 | 1.72 |
| 1 | J3 | CONN-SIL6 | 2751384 | 10 | 0.16 |
| 1 | L1 | HF50ACC635050 | 2292459 | 10 | 0.0983 |
| 1 | PWR | DC SOCKET DC8 | 2817542 | 10 | 0.694 |
| 1 | PWR LED | KA-3022SRC-4.5SF | 1142617 | 1 | 0.694 |
| 1 | RV1 | 5K | 652-3214G-1-502E | 1 | 2.78 |
| 1 | SP1 | LD1117S33TR | 1202826 | 5 | 0.341 |
| 2 | SW1, SW3 | SW-SPDT | 2435104 | 5 | 0.317 |
| 2 | SW2, SW4 | SW-DIP4 | 2452340 | 5 | 0.548 |
| 1 | SW5 | PVB4 OA | 1181018 | 5 | 0.712 |
| 1 | USB1 | UE27AC54100 | 2709068 | 10 | 0.349 |
| 2 | VR1, VR2 | S07K250 | 1004276 | 1 | 0.38 |
| 1 | X1 | 9C-8.000MEEJ-T | 2393160 | 1 | 0.59 |

## D. Capacidades de PCB (*PCBWay*)

Este anexo contiene la información necesaria para la fabricación de la PCB por la empresa *PCBWay*, donde se indican todos los parámetros referentes a la capacidad para fabricar la misma, y que coinciden con las reglas de diseño que colocamos en el Software de diseño antes de comenzar el proceso de desarrollo del *layout*.

| Items | | Manufacturing Capabilities | Remarks |
|---|---|---|---|
| Solder Mask |  | Green ,Red, Yellow, Blue, White ,Black | No extra charge (Green, Red, Yellow, Blue) |
| Silkscreen |  | White, Black, None | No extra charge. |
| Panelization |  | V-scoring,<br>Tab-routing,<br>Tab-routing with Perforation (Stamp Holes) | Leave min clearance of 1.6mm between boards for break-routing. For V-score panelization, set the space between boards to be zero. |
| Others | - | Fly Probe Testing (Free) and A.O.I. testing(free), ISO 9001:2008 ,UL Certificate | No extra charge. |

## Capacidades de PCB estándar - PCB estándar

| Categories | No. | Items | PCB process parameters | | | | Remarks |
|---|---|---|---|---|---|---|---|
| | | | Normal process | Medium difficulty | High difficulty | | |
| | | | | | Non-standard review | Unable to make | |
| product type | 1 | Multilayer PCB Layers | 3L≤Layers≤16L | 18L≤Layers≤24L | ≥24L | | |
| | 2 | Blind and Buried Vias | HDI(1+1+....+N+.....+1+1) | Anylayer HDI | HDI(2+...+N...+2) | | If meet the requirements of 2, 6, and 19 at the same time, it is classified as a high requirement product (thickness to diameter ratio, copper thickness of hole) |

| | No. | Items | | PCB process parameters | | | | Remarks |
|---|---|---|---|---|---|---|---|---|
| | | | | Normal process | Medium difficulty | High difficulty | | |
| | | | | | | Non-standard review | Unable to make | |
| | 3 | Surface Coating | | HASL(+gold finger),immersion gold, Immersion Gold +gold fingers with hard gold,OSP (+gold finger with hard gold), Immersion Tin (+gold finger with hard gold) (Not two different surface finish),Immersion Tin | Local immersion gold (long or short gold fingers, segmented gold finger craft) | Exceed this range require unconventional production processes | | Partial immersion gold, thickness of gold or nickel reference to the thickness of the coating |
| | 4 | Board Material | | FR-4;aluminum,Rogers4 series + FR-4 mixed(The Prepreg is ShengYi brand and ROGERS4403 series);CEM-3、LianMao IT158/IT180A | Pure ROGERS4 series multi-layer board (Prepreg is 4450F),PTFE、aluminum+FR4、PTFE+FR4 | Exceed this range require unconventional production processes | Pure PTFE multi-layer board | Pure PTFE can’t be made because the lamination temperature isn’t up to standard,Can‘t laminate Rogers copper foil directly |
| Drills | 5 | Drill diameter | Nc drill | 0.20mm≤Drill diameter≤6.5mm<br>More than 6.0mm using CNC milling<br>hole diameter 0.2mm: maximum board thickness 1.6mm<br>hole diameter diameter 0.25mm:maximum board thickness 2.0mm,<br>hole diameter 0.3mm≤Φ≤0.35mm, maximum board thickness 3.2 mm,<br>hole diameter 0.4mm≤Φ≤0.55mm, maximum board thickness 4.8 mm,<br>hole diameter>0.55mm maximum board thickness 6.4 mm | 6.5mm or more ±0.1mm ≤ hole diameter tolerance (using CNC milling for 6.5mm or more) | The drill diameter more than 6.0mm, the hole diameter tolerance less than ±0.1mm. If exceed this range require unconventional production processes | | Drill diameter below 0.2mm, and the aspect ratio≥10, which is medium difficulty |

| No. | Items | | | PCB process parameters | | | | Remarks |
|---|---|---|---|---|---|---|---|---|
| | | | | Normal process | Medium difficulty | High difficulty | | |
| | | | | | | Non-standard review | Unable to make | |
| 6 | Thickness to diameter ratio | | | Thickness to diameter ratio≤8 | 8 | 10 | Thickness to diameter ratio greater than 12 when the aperture cannot be compensated | If need to meet the requirement of 2, 6, and 19, it will be treat as high requirement product. |
| 7 | countersink | hole diameter | | 3.0mm≤hole diameter≤6.5mm | | Unconventional production beyond this range | | Countersink depth tolerance is controlled 0.15mm |
| | | Angle | | 90° | | Unconventional production beyond this range | | |
| 8 | Hole position tolerances | | | ±0.075mm | | ±0.05mm | <+ />-0.05mm | |
| 9 | hole diameter tolerance | PTH | | ±0.075mm or no customer requirements | ±0.05≤ hole diameter tolerance <±0.75mm | <±0.05mm | <+ />-0.05mm | Metallized hole diameter tolerance of 6.0mm or more refers to the requirement of serial number 5 |
| | | NPTH | | ≥±0.075mm | | <±0.05mm | <+ />-0.025mm | |
| | | Special hole | pressfit | ≤±0.05 | \ | \ | | |
| | | | non-plated Countersink/Counterbore holes(NPTH) | hole diameter <10mm:tolerance ±0.15mm,hole=±0.15mm,hole diameter=diameter ≥10mm:tolerance=≥10mm:tolerance ±=± 0.20mm=0.20mm | \ | \ | | |
| | | | non-plated Countersink/Counterbore holes（NPTH） | hole diameter <10mm:tolerance ±0.2mm=±0.2mm hole diameter ≥10mm:tolerance+0.3mm | \ | \ | | |

| No. | Items | | PCB process parameters | | | | Remarks |
|---|---|---|---|---|---|---|---|
| | | | Normal process | Medium difficulty | High difficulty | | |
| | | | | | Non-standard review | Unable to make | |
| 10 | Hole to hole spacing | component hole | ≥10MIL | 8≤Hole to hole spacing≤10 | 7≤Hole to hole spacing≤8 | <7mil | |
| | | via（≤0.45mm） | ≥8MIL | | | | |
| 11 | Slot (Cut-out) | Slot width | Plated slot ≥0.5mm<br>Non-plated slot ≥0.8mm | \ | \ | | More than 1.0mm, can be slot by machine |
| | | Length to width ratio of slot | Length to width≥2 | Length to width<2 | | | |
| 12 | Castellated Holes | Castellated Holes diameter | ≥0.5mm | 0.5mm>diameter≥0.4mm | \ | | |
| | | Castellated Holes spacing (edge to edge) | ≥0.3mm | 0.3mm>diameters≥0.2mm | \ | | |
| 13 | Minimum isolation ring of Inner layer, The distance between minimum hole in Inner layer and circuit (before compensation) | 4L | ≥7MIL | 6MIL≤isolation ring, distance<7mil | 5MIL≤isolation ring, distance<6mil | | If the size of one side is greater than 600MM, the inner hole to line and the hole to copper spacing must be greater than or equal to 15mil. If less than 15mil, it must be treated as unconventional review. The conventional process of 10 layers or more need to be incremented by 1 mil for each additional 2 |
| | | 6L | ≥8MIL | 6.5MIL≤isolation ring, distance<8mil | 6MIL≤isolation ring, distance<6.5mil | | |
| | | 8L | ≥9MIL | 7MIL≤isolation ring, distance<9mil | 6MIL≤isolation ring, distance<7mil | | |

| Categories | | | Items | | PCB process parameters | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | Normal process | Medium difficulty | High difficulty | | |
| | | | | | | | Non-standard review | Unable to make | |
| | | | ≥10L | | ≥10MIL | 8MIL≤isolation ring, distance<10mil<9MIL | 7MIL≤isolation ring, distance<8mil | | layers. Change the isolation ring to 12mil or more as much as possible |
| image transfer | 14 | The min width/spacing of inner layer (before compensation) | cooper thickness 18um | | ≥4/4 mil | ≥4/3.5 mil | | <3.5 />3 mil | width/spacing |
| | | | cooper thickness 35um | | ≥4/5 mil | ≥4/4 mil | | <3.5 />4 mil | width/spacing |
| | | | cooper thickness 70um | | ≥6/8mil | ≥6/7mil | | <5 />6 mil | width/spacing |
| | | | cooper thickness 105um | | ≥8/11 mil | ≥8/10 mil | | <6 />9 mil | width/spacing |
| | 15 | The min width/spacing of outer layer (before compensation) | cooper thickness 18um | | ≥4/5 mil | ≥4/4 mil or parts 3.5/3.5mil | | <3.5 />3.5 mil | Local 3.5/3.5mil, only the distance from the GBA chip area line to the PAD |
| | | | cooper thickness 35um | | ≥5/6 mil | ≥5/5 mil | | <4 />4 mil | |
| | | | cooper thickness 70um | | ≥7/8mil | ≥6/7mil | | <5 />6 mil | |
| | | | cooper thickness 105um | | ≥10/12 mil | ≥8/10 mil | | <6 />9 mil | |
| | 16 | grid trace width/spacing | cooper thickness 18um | | ≥7/9 mil | ≥6/8 mil | | <6 />7 mil | |
| | | | cooper thickness 35um | | ≥9/11 mil | ≥8/10 mil | | <8 />9 mil | |
| | | | cooper thickness 70um | | ≥11/13mil | ≥10/12mil | | <10 />11 mil | |
| | | | cooper thickness 105um | | ≥13/15 mil | ≥12/14 mil | | <12 />13 mil | |
| | 17 | Minimum weld ring (outer layer) | cooper thickness 18um | via hole | ≥5mil | ≥4mil | <3 mil=mil | | |
| | | | | component hole | ≥8mil | ≥6mil | <6 mil=mil | | |

| | | | Items | | PCB process parameters | | | | Remarks |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | Normal process | Medium difficulty | High difficulty | | |
| | | | | | | | Non-standard review | Unable to make | |
| | | | cooper thickness 35um | via hole | ≥5mil | ≥4mil | <3 mil=mil | | |
| | | | | component hole | ≥10mil | ≥8mil | <8 mil=mil | | |
| | | | cooper thickness 70um | via hole | ≥7mil | ≥6mil | <5 mil=mil | | |
| | | | | component hole | ≥12mil | ≥10mil | <10 mil=mil | | |
| | | | cooper thickness 105um | via hole | ≥8mil | ≥6mil | <6 mil=mil | | |
| | | | | component hole | ≥14mil | ≥12mil | <12 mil=mil | | |
| | 18 | width tolerance | | | width tolerance:≥±20% | ±10%≤ width tolerance:<±20% | <±10% | | spacing must meet the requirements of 11 and 12, If width is greater than 15mil, controlled by ±2.5mil |
| | | BGA pad diameter | hot air leveling (original) | | ≥12MIL | ≥10MIL | | <8mil | |
| | | | immersion gold (original) | | diameter≥11mil | 8.0mil≤diameter<11.0mil | | <6mil | |
| | 19 | Line to board edge distance | CNC milling | | 0.25mm | 0.20mm | <0.20mm | | |
| | | SMT width | | | ≥12mil | ≥9mil | <9mil以下 | | <7mil,except the=the binding=binding board=board |
| Metal plating | 20 | Plating Thickness(μin) | Electroless Nickel-Immersion Gold,ENIG | Nickel thickness | 100-150 μin | 200 μin | | | |

| | | | | PCB process parameters | | | | Remarks |
|---|---|---|---|---|---|---|---|---|
| | | | | Normal process | Medium difficulty | High difficulty | | |
| | | | | | | Non-standard review | Unable to make | |
| | | | gold thickness | 1-8 μin | | | >8 μin | |
| | | Full board gold plating | Nickel thickness | 100-150 μin | | 200-500 μin | | Order center check the final price |
| | | | gold thickness | 1-10 μin | 10-50 μin | | >50 μin | |
| | | gold finger | Nickel thickness | 120-150 μin | | 200-400 μin | | |
| | | | gold thickness | 1-30 μin | 30-50 μin | | >50 μin | |
| 21 | Hole copper thickness (μm) | Through hole | | 18-25 μm | 30-50 μm | >50 μm | | If 2,6,19 is required to exist at the same time, it will be treated as high requirement. The thickness of the copper is 25-50UM, and the thickness of the copper is required to be 2-3OZ generally. |
| | | Blind hole (mechanical hole) | | 18-25 μm | 30-50 μm | >50 μm | | |
| | | Buried hole | | 15-25 μm | 30-50 μm | >50 μm | | |
| 22 | Bottom copper thickness | Inner and outer copper thickness (OZ) | | 0.5-4 | 4-6 | | >6 | |

| Categories | No. | Items | | | PCB process parameters | | | | Remarks |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | Normal process | Medium difficulty | High difficulty | | |
| | | | | | | | Non-standard review | Unable to make | |
| solder mask | 23 | solder mask | green solder mask opening（mil） | | ≥2mil | 1.5 | 1 | | 1mil is only concentrated in the BGA area. If the window can be enlarged, increase it as much as possible, but the maximum is 3mil |
| | | | green solder mask Bridge（mil） | cooper thicknesss<2oz | 4（spacing between ICs is 8 mil, green oil），variegated or black oil≥4.5mil | 3-4（spacing between ICs is 7-8 mil, green oil），variegated or black oil≥4mil | | | |
| | | | | cooper thicknesss≥2OZ | 5 | 4 | | | |
| | | | Plug Hole diameter | | 0.20mm≤hole diameter≤0.40mm，plug hole fullness 70% | 0.4mm< hole diameter ≤0.70mm | fullness 100% | | |
| | | | Plug Hole board thickness | | 0.40mm≤board thickness≤2.4mm | >2.4MM | | | |
| | 24 | solder mask | solder mask color | | Green, matt green, blue, red, black, matte black, white, yellow | \ | \ | | Special colors need to be purchased or deployed in advance |
| silkscreen | 25 | Etched silkscreen (finished copper thickness) | Copper thickness 18um | word width/word height | 8MIL/40MIL | 7MIL/35MIL | | | |
| | | | Copper thickness 35um | word width/word height | 9MIL/40MIL | 8MIL/35MIL | | | |

| | | | Items | | PCB process parameters | | | | Remarks |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | Normal process | Medium difficulty | High difficulty | | |
| | | | | | | | Non-standard review | Unable to make | |
| | | | Copper thickness 75um | word width/word height | 12MIL/60MIL | 10MIL/50MIL | | | |
| | | | Copper thickness 105um | word width/word height | 16MIL/60MIL | 14MIL/50MIL | | | |
| outline | 26 | Maximum board thickness | Double PCB | | 3.2MM | 4.5MM | >4.5MM | | calculated by 4 layers if the thickness more than 3mm |
| | | | Multilayer layer board | | 3.2MM | 4.5MM | >4.5MM | | |
| | 27 | Minimum board thickness (single and double panel refers to substrate thickness) | Single or Double side PCB (pcb prototype) | | ≥0.3mm | 0.25mm | | | |
| | | | 4L | | ≥0.60mm | 0.40mm | <0.40mm | | |
| | | | 6L | | ≥0.9mm | 0.70mm | <0.70mm | | |
| | | | 8L | | ≥1.20mm | 1.00mm | <1.00mm | | |
| | | | 10L | | ≥1.40mm | 1.20mm | <1.20mm | | |
| | | | 12L | | ≥1.70mm | 1.50mm | <1.50mm | | |
| | | | 14L | | ≥2.00mm | 1.80mm | <1.80mm | | |
| | 28 | thickness (T) tolerance MM (multilayer layer pcb) | T≤1.0 | | ±0.10 | | Need to review if less than the tolerance | | If the tolerance is unilateral tolerance, the tolerance shall be double tolerance value, such as: 1.8mm requires |
| | | | 1.0 | | ±0.13 | | | | |
| | | | 1.6 | | ±0.18 | | | | |
| | | | 2.5 | | ±0.23 | | | | |

| | | Items | PCB process parame | | | | |
|---|---|---|---|---|---|---|---|
| | | | Normal process | Medium difficulty | | | |
| | | T≥3.2 | ±8% | | | | positive tolerance, the tolerance shall be 0-0.36mm |
| 29 | Maximum finished board size | Single and double side PCB | 508×610mm | Beyond this range needs to be reviewed | | | |
| | | Multilayer Layer PCB | 508×600mm | | | | |
| 30 | Minimum finished pcb size | | ≥20mm | 10mm≤Size<20mm | <10mm | | |
| 31 | Beveling for gold finger | Bevel angle | 20°30°45°60° | | <20°or>60° | | |
| | | Bevel angle tolerance | >±5° | ±5° | <±5° | | |
| | | Bevel depth tolerance | tolerance≥±0.15mm | ±0.15mm< Tolerance ≤ ±0.1mm | tolerance<±0.10mm | | |
| 32 | Shape tolerance | | tolerance≥±0.15mm | ±0.10mm≤tolerance<±0.15mm | | Tolerance<±0.10mm or=or more=more than=than two=two form=form form=form form=form form=form form=form form=form form=form form=form tolerance=tolerance control=control | |
| 33 | V-CUT | Angle | 20°30°45°60° | | | | |
| | | The Maximum number of V-CUT | In 20 times | In 30 times | In 40 times | | |

| | | | Items | PCB process parameters | | | | Remarks |
|---|---|---|---|---|---|---|---|---|
| | | | | Normal process | Medium difficulty | High difficulty | | |
| | | | | | | Non-standard review | Unable to make | |
| | | | Width of the shape | 80MM< width <560mm | 60MM< width <80mm | width <60mm | | |
| | | | board thickness | 0.6MM≦thickness≦2.4MM | 0.5MM≦thickness<0.6mm | thickness<0.5mm or=or thickness=thickness>2.4MM | | below 0.5mm is single-sided V-CUT |
| | | | Remaining thickness | ≥0.25MM | | | <0.25mm | |
| | | | V-CUT | Conventional V-CUTT、V-CUT: Skip V-CUT | \ | \ | | |
| others | 34 | panel size | The minimum panel size | ≥100*120mm | \ | <100*120mm | | The thickness of the finished board is less than 0.4MM, the panel size can't exceed 14inch, and the maximum size of the HASL PCB can't exceed 24inch |
| | | | the Maximum panel size | ≤20*24 inch | \ | Need to review if beyond range | | |
| | 35 | impedance control | Impedance control tolerance | ±10%，50Ω and below：±5Ω | \ | <±10%，50ω and=and below=below <±5Ω | | |
| | | bow and twist | bow and twist tolerance | bow and twist≥0.75% | 0.75%≤bow and twist≤0.5% | bow and twist<0.5% | | asymmetry boards bow and twist tolerance 1.2% |
| | 36 | HASL processing capacity | component hole diameter | hole diameter>0.5mm | 0.4mm≤hole diameter≤0.5mm | | | |
| | | | board thickness | 0.5mm≤board thickness≤3.5mm | 0.4mm≤board thickness<0.5mm | | | |
| | | | thickness | 2um≤thickness of Tin≤30um | \ | \ | | |

# E. Código fuente del Firmware

Este anexo contiene el código, programado en lenguaje *C*, de los archivos principales que componen el programa cargado en el Hardware del sistema. No se incluyen las librerías genéricas del proyecto que no se hayan modificado.

```
 1 /*
 2  * Project: OscAndroid_Firmware
 3  * File:    main.c
 4  * Author:  David Ortiz de Latierro Delgado
 5  * Rev:     1.0
 6  */
 7
 8 // ****************************************************************************
 9 // ****************************************************************************
10 // Seccion: Archivos #include
11 // ****************************************************************************
12 // ****************************************************************************
13
14 #include <system.h>
15 #include <usb.h>
16 #include <usb_host_android.h>
17 #include <src/app_android_basic_io.h>
18
19 // ****************************************************************************
20 // ****************************************************************************
21 // Seccion: Definicion de macros
22 // ****************************************************************************
23 // ****************************************************************************
24
25 #ifndef MAX_ALLOWED_CURRENT
26     #define MAX_ALLOWED_CURRENT     (500)   // Máxima corriente permitida en mA
27 #endif
28
29 /*****************************************************************************
30  * Funcion:         int main(void);
31  *
32  * Resumen:         Programa principal
33  *
34  * Descripcion:     Funcion principal del programa
35  *
36  * Parametros:      Ninguno
37  *
38  * Return:          int (0 por defecto)
39  *
40  * Notas:           Nada
41  *****************************************************************************/
42
43 int main(void)
44 {
45     // Inicializamos el sistema
46     init();
```

```
47
48      // Inicializamos los modulos ADC
49      initAdc1();
50      initAdc2();
51
52      // Inicializamos la libreria USB Host Android
53      USBInitialize(0);           // Inicializa el stack de USB Host
54      App_Android_Initialize();   // Configura la informacion del accesorio USB,
55                                  // e inicializa la informacion del driver del
56                                  // dispositivo Android
57      while(1)
58      {
59          USBTasks();         // Seguimos ejecutando el stak de USB y Android
60          AndroidTasks();
61
62          App_Android_Tasks();// Ejecutamos el ciclo principal de nuestro programa
63      }
64  }
65
66  /*********************************************************************
67   * Funcion:         bool USB_ApplicationDataEventHandler(uint8_t address,
68   *                  USB_EVENT event, void *data, uint32_t size);
69   *
70   * Resumen:         Funcion de respuesta a datos del stack de USB
71   *
72   * Descripcion:     Esta funcion es llamada cada vez que el stack de USB quiere
73   *                  responder a un dato recibido, permitiendo al usuario
74   *                  tomar decisiones clave sobre el stack durante el tiempo de
75   *                  ejecucion
76   *
77   * Parametros:      Ninguno
78   *
79   * Return:          Nada
80   *
81   * Notas:           Nada
82   ********************************************************************/
83
84  bool USB_ApplicationDataEventHandler(uint8_t address, USB_EVENT event,
85          void *data, uint32_t size)
86  {
87      return false;
88  }
89
90  /*********************************************************************
91   * Funcion:         bool USB_ApplicationEventHandler(uint8_t address,
92   *                  USB_EVENT event, void *data, uint32_t size);
```

```
 93  *
 94  * Resumen:         Funcion de respuesta a eventos del stack de USB
 95  *
 96  * Descripcion:     Esta funcion es llamada cada vez que el stack de USB quiere
 97  *                  notificar al usuario de un evento, permitiendo al usuario
 98  *                  tomar decisiones clave sobre el stack durante el tiempo de
 99  *                  ejecucion
100  *
101  * Parametros:      Ninguno
102  *
103  * Return:          Nada
104  *
105  * Notas:           Nada
106  ********************************************************************************/
107
108 bool USB_ApplicationEventHandler(uint8_t address, USB_EVENT event,
109         void *data, uint32_t size)
110 {
111     switch(event)
112     {
113         case EVENT_VBUS_REQUEST_POWER:
114             // Si el dispositivo demanda demasiada corriente, se rechaza.
115             if (((USB_VBUS_POWER_EVENT_DATA*)data)->current <=
116                     (MAX_ALLOWED_CURRENT / 2))
117             {
118                 return true;
119             }
120             else
121             {
122             }
123             break;
124
125         case EVENT_VBUS_RELEASE_POWER:
126         case EVENT_HUB_ATTACH:
127         case EVENT_UNSUPPORTED_DEVICE:
128         case EVENT_CANNOT_ENUMERATE:
129         case EVENT_CLIENT_INIT_ERROR:
130         case EVENT_OUT_OF_MEMORY:
131         case EVENT_UNSPECIFIED_ERROR:   // Este nunca deberia generarse
132         case EVENT_DETACH:              // Se ha extraido el cable USB
133
134         case EVENT_ANDROID_DETACH:
135             App_Android_Detach(data);
136             return true;
137             break;
138
```

```
139         case EVENT_ANDROID_ATTACH:
140             App_Android_Attach(data);
141             return true;
142 
143         default :
144             break;
145     }
146     return false;
147 }
148 
149 /*****************************************************************************
150  Fin de archivo
151 */
```

```
 1 /*
 2  * Project: OscAndroid_Firmware
 3  * File:    app_android.c
 4  * Author:  David Ortiz de Latierro Delgado
 5  * Rev:     1.0
 6  */
 7
 8 // ****************************************************************************
 9 // ****************************************************************************
10 // Seccion: Archivos #include
11 // ****************************************************************************
12 // ****************************************************************************
13
14 #include <stdint.h>
15
16 #include <usb.h>
17 #include <usb_host_android.h>
18
19 #include <system.h>
20
21 #include "app_android_basic_io.h"
22
23 // ****************************************************************************
24 // ****************************************************************************
25 // Seccion: Definicion de macros
26 // ****************************************************************************
27 // ****************************************************************************
28
29 #define COMANDO_XMAX 0xFF   // Comando que indica inicio de transmision de Xmax
30 #define I_AM_READY   0xF5   // Comando que indica que la App esta lista
31 #define PINTADO      0xF3   // Comando que indica el pintado de una pantalla
32
33 // ****************************************************************************
34 // ****************************************************************************
35 // Seccion: Definicion de variables
36 // ****************************************************************************
37 // ****************************************************************************
38
39 void* device_handle = NULL;
40 static bool device_attached = false;
41
42 static bool writeInProgress = false;
43 static bool readInProgress = false;
44
45 static uint32_t transferSize = 0;
46 static uint8_t errorCode;
```

```
47 static uint8_t comando[2];
48
49 static char manufacturer[] = "Universidad de Valladolid";
50 static char model[] = "OscAndroid";
51 static char description[] = "TFG Grado en Ing. de Tecnol. Espec. de "
52 "Telecomunicacion";
53 static char version[] = "1.0";
54 static char uri[] = "FALTA URL DE LA APP EN PLAY STORE";
55 static char serial[] = "N/A";
56
57 ANDROID_ACCESSORY_INFORMATION myDeviceInfo =
58 {
59     manufacturer,
60     sizeof(manufacturer),
61     model,
62     sizeof(model),
63     description,
64     sizeof(description),
65     version,
66     sizeof(version),
67     uri,
68     sizeof(uri),
69     serial,
70     sizeof(serial)
71 };
72
73 static bool inicio_programa = true;
74 static bool datos_listos = false;
75 static bool need_update = true;
76 static bool nueva_pantalla = true;
77 static bool primer_valor = true;
78 static bool app_ready = false;
79
80 static int valor1 = 0;
81 static int valor2 = 0;
82 static int valor_trigger = 0;
83 static uint8_t trigger = 0x00;
84 static uint8_t trigger_before = 0xF0;
85
86 static int xmax = 0;
87 static int escala_tiempos = 0;
88 static unsigned short tasa_envio[15];
89 static long num_muestras[15] = {100,200,500,1000,2000,5000,10000,20000,50000,
90                                 1e5,2e5,5e5,1e6,2e6,5e6};
91
92 static uint8_t read_buffer[64];
```

```
 93 static uint8_t buffer_salida[8000];
 94
 95 static int i = 0;
 96 static int j = 0;
 97 static int k = 0;
 98
 99 static bool pintado = false;
100
101 /*****************************************************************************
102  * Funcion:         void App_Android_Initialize(void);
103  *
104  * Resumen:         Inicializacion driver cliente USB
105  *
106  * Descripcion:     Configura la informacion del accesorio USB, e inicializa la
107  *                  informacion del driver del dispositivo Android.
108  *
109  * Parametros:      Ninguno
110  *
111  * Return:          Nada
112  *
113  * Notas:           Nada
114  *****************************************************************************/
115
116 void App_Android_Initialize(void)
117 {
118     writeInProgress = false;
119     readInProgress = false;
120     AndroidAppStart(&myDeviceInfo); // Definimos la informacion de nuestro
121                                     // accesorio para el dispositivo Android
122     device_attached = false;
123     app_ready = false;
124     inicio_programa = true;
125 }
126
127 /*****************************************************************************
128  * Funcion:         void App_Android_Tasks(void);
129  *
130  * Resumen:         Ciclo de programa principal
131  *
132  * Descripcion:     Es el ciclo que define el funcionamiento principal del
133  *                  programa para enviar y recibir comandos y/o datos.
134  *
135  * Parametros:      Ninguno
136  *
137  * Return:          Nada
138  *
```

```
139  * Notas:           Nada
140  ***************************************************************************/
141
142 void App_Android_Tasks(void)
143 {
144     // Si el dispositivo aun no se ha conectado, o ha sufrido una desconexion
145     if(device_attached == false)
146     {
147         // Detenemos los procesos de captura de los ADC
148         AD1CON1bits.ADON = 0;
149         AD2CON1bits.ADON = 0;
150         // Reseteamos las variables de ejecucion del programa
151         app_ready = false;
152         inicio_programa = true;
153         datos_listos = false;
154         need_update = true;
155         nueva_pantalla = true;
156         primer_valor = true;
157         trigger_before = 0xF0;
158         transferSize = 0;
159         xmax = 0;
160         trigger = 0x00;
161         escala_tiempos = 0;
162         i = 0;
163         j = 0;
164         k = 0;
165
166         return; // Salimos al bucle while (en main.c) para comprobar de nuevo
167     }
168
169     //***************************************************************************
170     // Si el dispositivo esta conectado, ejecutamos el ciclo de programa
171     if(readInProgress == false)
172     {
173         errorCode = AndroidAppRead(device_handle, (uint8_t*)&read_buffer,
174                 (uint32_t)sizeof(read_buffer));
175         // Si el dispositivo esta conectado esperamos a una respuesta
176         if( errorCode != USB_SUCCESS)
177         {
178             //Error
179         }
180         else // Recibimos una respuesta a un comando
181         {
182             readInProgress = true;
183         }
184     }
```

```
185
186     //************************************************************************
187     transferSize = 0;
188
189     if(AndroidAppIsReadComplete(device_handle, &errorCode,
190             &transferSize) == true)
191     {
192         if(errorCode == USB_SUCCESS)    // Hemos recibido informacion del
193         {                               // dispositivo Android
194
195             AD1CON1bits.ADON = 0; // Detiene el ADC1
196             AD2CON1bits.ADON = 0; // Detiene el ADC2
197
198             while(transferSize > 0) // Comprobamos de que informacion se trata
199             {
200                 if(app_ready == false)
201                 {
202                     if(read_buffer[0] == I_AM_READY)
203                     {
204                         app_ready = true; // App lista para enviar/recibir datos
205                         transferSize -= 1;
206                     }
207                 }
208                 else
209                 {
210                     switch(read_buffer[0])
211                     {
212                         case COMANDO_XMAX:    // Nos envian el valor de Xmax
213                             xmax = (int)(read_buffer[1] & 0xFF);
214                             xmax *= 256;
215                             xmax += (int)(read_buffer[2] & 0xFF);
216
217                             for(i=0;i<15;i++)
218                             {
219                                 if(num_muestras[i] <= xmax)
220                                 {
221                                     tasa_envio[i] = 1;
222                                 }
223                                 else
224                                 {
225                                     tasa_envio[i] = (unsigned short)
226                                             (num_muestras[i]/xmax);
227                                 }
228                             }
229                             transferSize -= 3;
230                         break;
```

```
231
232                         case PINTADO:     // Nos avisan para mandar mas muestras
233                             read_buffer[0] = read_buffer[1];
234                             pintado = false;
235                             transferSize -= 1;
236                         break;
237
238                         default:            // Nos envian los flags de estado
239                             trigger = read_buffer[0] & 0x0C;
240
241                             escala_tiempos = (int)read_buffer[0] & 0xF0;
242                             escala_tiempos /= 16;
243
244                             datos_listos = false;
245                             nueva_pantalla = true;
246                             primer_valor = true;
247                             i = 0;
248                             j = 0;
249                             k = 0;
250
251                             if((read_buffer[0] & 0x01) == 0)
252                             {
253                                 AD1CON1bits.ADON = 0;   // Canal 1 apagado
254                             }
255                             else
256                             {
257                                 AD1CON1bits.ADON = 1;   // Canal 1 encendido
258                             }
259                             if((read_buffer[0] & 0x02) == 0)
260                             {
261                                 AD2CON1bits.ADON = 0;   // Canal 2 apagado
262                             }
263                             else
264                             {
265                                 AD2CON1bits.ADON = 1;   // Canal 2 encendido
266                             }
267
268                             if((AD1CON1bits.ADON == 0) &&
269                                     (AD2CON1bits.ADON == 0))
270                             {
271                                 need_update = true;
272                             }
273                             else
274                             {
275                                 need_update = false;
276                             }
```

```
277
278                             transferSize -= 1;
279                         break;
280                     }
281                 }
282             }
283         }
284         else
285         {
286             //Error
287         }
288     }
289
290     //*********************************************************************
291     // Si ya no nos queda nada por leer...
292     if(transferSize == 0)
293     {
294         readInProgress = false;
295     }
296
297     //*********************************************************************
298     // Si ya tenemos todos los puntos para representar en el buffer de salida
299     // indicamos que los datos estan listos para ser enviados
300     if(xmax != 0)
301     {
302         if(j > ((xmax-1)*4))
303         {
304             AD1CON1bits.ADON = 0; // Detiene el ADC1
305             if(AD2CON1bits.ADON == 1)
306             {
307                 if(k > (((xmax-1)*4)+2))
308                 {
309                     AD2CON1bits.ADON = 0; // Detiene el ADC2
310                     datos_listos = true;
311                 }
312                 else
313                 {
314                     datos_listos = false;
315                 }
316             }
317             else
318             {
319                 datos_listos = true;
320             }
321         }
322
```

```
323         if((k > (((xmax-1)*4)+2)) && (datos_listos == false))
324         {
325             AD2CON1bits.ADON = 0; // Detiene el ADC2
326             if(AD1CON1bits.ADON == 1)
327             {
328                 if(j > ((xmax-1)*4))
329                 {
330                     AD1CON1bits.ADON = 0; // Detiene el ADC1
331                     datos_listos = true;
332                 }
333                 else
334                 {
335                     datos_listos = false;
336                 }
337             }
338             else
339             {
340                 datos_listos = true;
341             }
342         }
343     }
344 
345     //*********************************************************************
346     // Si hay una escritura en progreso, necesitamos comprabar si ha terminado
347     if(writeInProgress == true)
348     {
349         if(AndroidAppIsWriteComplete(device_handle, &errorCode,
350                 &transferSize) == true)
351         {
352             writeInProgress = false;
353 
354             if(errorCode != USB_SUCCESS)
355             {
356                 //Error
357             }
358         }
359     }
360 
361     //*********************************************************************
362     // Si la App aun no esta disponible, esperamos sin hacer nada mas
363     if(app_ready == false)
364     {
365         return;
366     }
367 
368     //*********************************************************************
```

```
369     // Si llegamos a este punto es porque la App ya esta lista y tiene datos
370     // validos que poder enviarnos
371
372     if((inicio_programa == true) && (writeInProgress == false))
373     {
374         // Le solicitamos al dispositivo que nos envie el valor de Xmax
375         comando[0] = 0xFF;
376         comando[1] = 0xFF;
377
378         errorCode = AndroidAppWrite(device_handle,(uint8_t*)&comando, 2);
379         if( errorCode != USB_SUCCESS )
380         {
381             //Error
382         }
383         else
384         {
385             inicio_programa = false;
386             writeInProgress = true;
387         }
388     }
389
390     //*************************************************************************
391     // Le solicitamos el estado de los controles de la App
392     if((need_update == true) && (inicio_programa == false) &&
393             (writeInProgress == false))
394     {
395         comando[0] = 0xFF;
396         comando[1] = 0xF0;
397
398         errorCode = AndroidAppWrite(device_handle,(uint8_t*)&comando, 2);
399         if( errorCode != USB_SUCCESS )
400         {
401             //Error
402         }
403         else
404         {
405             need_update = false;
406             writeInProgress = true;
407         }
408     }
409
410     //*************************************************************************
411     // Enviamos el buffer de datos hacia el dispositivo Android
412     if((datos_listos == true) && (writeInProgress == false) &&
413             (pintado == false))
414     {
```

```
415         errorCode = AndroidAppWrite(device_handle,
416                 (uint8_t*)&buffer_salida, (xmax*4));
417         if( errorCode != USB_SUCCESS )
418         {
419             //Error
420         }
421         else
422         {
423             pintado = true;
424             datos_listos = false;
425             j = 0;
426             k = 0;
427             need_update = true;
428             writeInProgress = true;
429         }
430     }
431 }
432 
433 /*****************************************************************************
434  * Funcion:         void _ISR _ADC1Interrupt(void);
435  *
436  * Resumen:         Rutina de interrupcion para el ADC 1
437  *
438  * Descripcion:     Es la rutina de servicio de interrupcion (ISR) del ADC 1,
439  *                  la cual se activa cada vez que termina una conversion.
440  *
441  * Parametros:      Ninguno
442  *
443  * Return:          Nada
444  *
445  * Notas:           Nada
446  *****************************************************************************/
447 
448 void _ISR _AD1Interrupt(void)
449 {
450     IFS0bits.AD1IF = 0;     // Borramos el flag de notificacion de interrupcion
451 
452     if(j > ((xmax-1)*4))    // Ya tenemos todas las muestras para representar
453     {                       // una pantalla
454         AD1CON1bits.ADON = 0; // Detiene el ADC1
455         //return;
456     }
457 
458     if(nueva_pantalla == true)      // Aplicamos el algoritmo para el trigger
459     {
460         if((trigger & 0x08) == 0)       // Comprobamos si es el canal elegido
```

```
461         {
462             if(primer_valor == true)    // Comprobamos si es el primer valor
463             {                           // que capturamos
464                 valor1 = ADC1BUF0;
465                 primer_valor = false;
466             }
467             else
468             {
469                 valor2 = ADC1BUF0;
470                 if(trigger != trigger_before)   // Comprobamos si ya tenemos un
471                 {                               // valor para el trigger
472                     trigger_before = trigger;
473                     valor_trigger = valor2;
474                 }
475                 if((trigger & 0x04) == 0)   // Flanco de subida
476                 {
477                     if((valor1 < valor_trigger) && (valor_trigger <= valor2))
478                     {
479                         nueva_pantalla = false;
480                         i = 0;
481                         j = 0;
482                         buffer_salida[j]   = (uint8_t) 0x00FF & (ADC1BUF0 >> 8);
483                         buffer_salida[j+1] = (uint8_t) 0x00FF & ADC1BUF0;
484                         j += 4;
485                     }
486                     else
487                     {
488                         valor1 = valor2;
489                     }
490                 }
491                 else                        // Flanco de bajada
492                 {
493                     if((valor1 > valor_trigger) && (valor_trigger >= valor2))
494                     {
495                         nueva_pantalla = false;
496                         i = 0;
497                         j = 0;
498                         buffer_salida[j]   = (uint8_t) 0x00FF & (ADC1BUF0 >> 8);
499                         buffer_salida[j+1] = (uint8_t) 0x00FF & ADC1BUF0;
500                         j += 4;
501                     }
502                     else
503                     {
504                         valor1 = valor2;
505                     }
506                 }
```

```
507             }
508         }
509     }
510     else
511     {
512         i++;
513         if(i == tasa_envio[escala_tiempos])
514         {
515             buffer_salida[j]   = (uint8_t) 0x00FF & (ADC1BUF0 >> 8);
516             buffer_salida[j+1] = (uint8_t) 0x00FF & ADC1BUF0;
517             j += 4;
518 
519             i = 0;
520         }
521     }
522 }
523 
524 /*****************************************************************************
525  * Funcion:         void _ISR _ADC2Interrupt(void);
526  *
527  * Resumen:         Rutina de interrupcion para el ADC 2
528  *
529  * Descripcion:     Es la rutina de servicio de interrupcion (ISR) del ADC 2,
530  *                  la cual se activa cada vez que termina una conversion.
531  *
532  * Parametros:      Ninguno
533  *
534  * Return:          Nada
535  *
536  * Notas:           Nada
537  *****************************************************************************/
538 
539 void _ISR _AD2Interrupt(void)
540 {
541     IFS1bits.AD2IF = 0; // Borramos el flag de notificacion de interrupcion
542 
543     if(k > (((xmax-1)*4)+2))
544     {
545         //AD2CON1bits.ADON = 0; // Detiene el ADC2
546         return;
547     }
548 
549     if(nueva_pantalla == true)  // Aplicamos el algoritmo para el trigger
550     {
551         if((trigger & 0x08) == 8)  // Comprobamos si es el canal elegido
552         {
```

```
553             if(primer_valor == true)    // Comprobamos si es el 1er valor
554             {                           // que capturamos
555                 valor1 = ADC2BUF0;
556                 primer_valor = false;
557             }
558             else
559             {
560                 valor2 = ADC2BUF0;
561                 if(trigger != trigger_before)   // Comprobamos si ya tenemos un
562                 {                               // valor para el trigger
563                     trigger_before = trigger;
564                     valor_trigger = valor2;
565                 }
566                 if((trigger & 0x04) == 0)   // Flanco de subida
567                 {
568                     if((valor1 < valor_trigger) && (valor_trigger <= valor2))
569                     {
570                         nueva_pantalla = false;
571                         i = 0;
572                         k = 2;
573                         buffer_salida[k]   = (uint8_t) 0x00FF & (ADC2BUF0 >> 8);
574                         buffer_salida[k+1] = (uint8_t) 0x00FF & ADC2BUF0;
575                         k += 4;
576                     }
577                     else
578                     {
579                         valor1 = valor2;
580                     }
581                 }
582                 else                        // Flanco de bajada
583                 {
584                     if((valor1 > valor_trigger) && (valor_trigger >= valor2))
585                     {
586                         nueva_pantalla = false;
587                         i = 0;
588                         k = 2;
589                         buffer_salida[k]   = (uint8_t) 0x00FF & (ADC2BUF0 >> 8);
590                         buffer_salida[k+1] = (uint8_t) 0x00FF & ADC2BUF0;
591                         k += 4;
592                     }
593                     else
594                     {
595                         valor1 = valor2;
596                     }
597                 }
598             }
```

```
599         }
600     }
601     else
602     {
603        i++;
604        if(i == tasa_envio[escala_tiempos])
605        {
606            buffer_salida[k]   = (uint8_t) 0x00FF & (ADC2BUF0 >> 8);
607            buffer_salida[k+1] = (uint8_t) 0x00FF & ADC2BUF0;
608            k += 4;
609
610            i = 0;
611        }
612     }
613 }
614
615 /*****************************************************************************
616  * Funcion:         App_Android_Detach(void* handle);
617  *
618  * Resumen:         Respuesta a evento de desconexion.
619  *
620  * Descripcion:     Identifica el dispositivo como desconectado y establece
621  *                  el ciclo inicial del programa.
622  *
623  * Parametros:      void* handle
624  *
625  * Return:          Nada
626  *
627  * Notas:           Nada
628  *****************************************************************************/
629
630 void App_Android_Detach(void* handle)
631 {
632     device_attached = false;
633     inicio_programa = true;
634     device_handle = NULL;
635 }
636
637 /*****************************************************************************
638  * Funcion:         void App_Android_Attach(void* handle);
639  *
640  * Resumen:         Respuesta a evento de conexion.
641  *
642  * Descripcion:     Identifica el dispositivo como conectado.
643  *
644  * Parametros:      void* handle
```

```
645  *
646  * Return:          Nada
647  *
648  * Notas:           Nada
649  ******************************************************************************/
650
651 void App_Android_Attach(void* handle)
652 {
653     device_attached = true;
654     device_handle = handle;
655 }
656
657 /*******************************************************************************
658  Fin de archivo
659 */
```

```
1 /*
2  * Proyecto:    OscAndroid_Firmware
3  * Archivo:     app_android.h
4  * Autor:       David Ortiz de Latierro Delgado
5  * Rev:         1.0
6  */
7
8 // ***************************************************************************
9 // ***************************************************************************
10 // Seccion: Declaraciones de las funciones
11 // ***************************************************************************
12 // ***************************************************************************
13
14 void App_Android_Initialize(void);
15 void App_Android_Tasks(void);
16 void App_Android_Detach(void* handle);
17 void App_Android_Attach(void* handle);
18
19 /***************************************************************************
20  Fin de archivo
21 */
```

```
 1 /*
 2  * Proyecto:    OscAndroid_Firmware
 3  * Archivo:     system.c
 4  * Autor:       David Ortiz de Latierro Delgado
 5  * Rev:         1.0
 6  */
 7
 8 // ****************************************************************************
 9 // ****************************************************************************
10 // Seccion: Archivos #include
11 // ****************************************************************************
12 // ****************************************************************************
13
14 #include <xc.h>
15 #include <system.h>
16 #include <usb.h>
17 #include <stdint.h>
18 #include <stdbool.h>
19
20 // ****************************************************************************
21 // ****************************************************************************
22 // Seccion: Palabra de configuracion para DSPIC33EP256MU806
23 // ****************************************************************************
24 // ****************************************************************************
25
26 // FGS
27 #pragma config GWRP = OFF           // General Segment Write-Protect bit
28 #pragma config GSS = OFF            // General Segment Code-Protect bit
29 #pragma config GSSK = OFF           // General Segment Key bits
30
31 // FOSCSEL
32 #pragma config FNOSC = FRC          // Initial Oscillator Source Selection bits
33 #pragma config IESO = OFF           // Two-speed Oscillator Start-up Enable bit
34
35 // FOSC
36 #pragma config POSCMD = HS          // Primary Oscillator Mode Select bits
37 #pragma config OSCIOFNC = OFF       // OSC2 Pin Function bit
38 #pragma config IOL1WAY = OFF        // Peripheral pin select configuration
39 #pragma config FCKSM = CSECMD       // Clock Switching Mode bits
40
41 // FWDT
42 #pragma config WDTPOST = PS32768    // Watchdog Timer Postscaler bits (1:32,768)
43 #pragma config WDTPRE = PR128       // Watchdog Timer Prescaler bit (1:128)
44 #pragma config PLLKEN = ON          // PLL Lock Wait Enable bit
45 #pragma config WINDIS = OFF         // Watchdog Timer Window Enable bit
46 #pragma config FWDTEN = OFF         // Watchdog Timer Enable bit
```

```
47
48 // FPOR
49 #pragma config FPWRT = PWR128       // Power-on Reset Timer Value Select bits
50 #pragma config BOREN = OFF          // Brown-out Reset(BOR) Detection Enable bit
51 #pragma config ALTI2C1 = OFF        // Alternate I2C pins for I2C1
52
53 // FICD
54 #pragma config ICS = PGD1           // ICD Communication Channel Select bits
55 #pragma config RSTPRI = PF          // Reset Target Vector Select bit
56 #pragma config JTAGEN = OFF         // JTAG Enable bit
57
58 // FAS
59 #pragma config AWRP = OFF           // Auxiliary Segment Write-protect bit
60 #pragma config APL = OFF            // Auxiliary Segment Code-protect bit
61 #pragma config APLK = OFF           // Auxiliary Segment Key bits
62
63 // ******************************************************************************
64 // ******************************************************************************
65 // Seccion: Definicion de constantes
66 // ******************************************************************************
67 // ******************************************************************************
68
69 #define Fosc    120000000
70 #define Fcy     (Fosc/2)
71 #define Fin     14.7456e6
72
73 /*********************************************************************************
74  * Funcion:         void init(void);
75  *
76  * Resumen:         Funcion que inicializa el sistema
77  *
78  * Descripcion:     Funcion para inicializar el reloj del sistema, configurar
79  *                  el PLL y el PLL auxiliar y definir los puertos
80  *
81  * Parametros:      Ninguno
82  *
83  * Return:          Nada
84  *
85  * Notas:           Se debe configurar el PLL segun la frecuencia de entrada
86  *********************************************************************************/
87
88 void init(void)
89 {
90     // Configuramos el OSC para trabajar a 60 MIPS con un cristal de 14.7456 MHz
91     // Fosc = (Fin*M)/(N1*N2) --> Fosc = (14.7456e6*49)/(3*2) = 120,42MHz
92     PLLFBD = 47;            // M  = 49 --> M  = PLLDIV + 2
```

```
 93     CLKDIVbits.PLLPOST = 0; // N2 =  2 --> N2 = 2*(PLLPOST + 1)
 94     CLKDIVbits.PLLPRE = 1;  // N1 =  3 --> N1 = PLLPRE + 2
 95
 96     // Iniciamos el cambio de fuente de reloj al oscilador principal con PLL
 97     __builtin_write_OSCCONH(0x03);
 98     __builtin_write_OSCCONL(OSCCON | 0x01);
 99
100     // Esperamos a que cambie el reloj
101     while(OSCCONbits.COSC!= 0b011);
102
103     // Esperamos a que el PLL se fije
104     while(OSCCONbits.LOCK!= 1);
105
106     // Configuracion del PLL para el modulo USB, que trabaja a 48MHz
107     // Aclk = (Fin*M)/(N1*N2) --> Aclk = (14.7456e6*20)/(3*2) = 49,15MHz
108     ACLKCON3bits.ASRCSEL = 1;   // Oscilador primario como fuente de reloj
109     ACLKCON3bits.FRCSEL = 0;    // Oscilador primario como fuente de reloj
110     ACLKCON3bits.SELACLK = 1;   // Seleccionar PLL auxiliar para proveer la
111                                 // fuente de reloj para el divisor auxiliar
112     ACLKDIV3bits.APLLDIV = 0b101;    // M = 20 --> M  = APLLDIV  + 15
113     ACLKCON3bits.APLLPRE = 0b010;    // N1 = 3 --> N1 = APLLPRE  +  1
114     ACLKCON3bits.APLLPOST = 0b110;   // N2 = 2
115
116     ACLKCON3bits.ENAPLL = 1;    // Habilitamos reloj auxiliar
117
118     // Configuramos los puertos y definimos las entradas analogicas
119     TRISB = TRISC = TRISD = TRISE =
120     TRISG = 0;              // Todo como salidas
121     TRISBbits.TRISB0 = 1;   // AN0(RB0) y AN1(RB1) como entradas
122     TRISBbits.TRISB1 = 1;
123
124     ANSELB = ANSELC = ANSELD = ANSELE =
125     ANSELG = 0x0000;        // Todas las lineas digitales
126     ANSELBbits.ANSB0 = 1;   // AN0/RB0 entrada analogica para el ADC 1
127     ANSELBbits.ANSB1 = 1;   // AN1/RB1 entrada analogica para el ADC 2
128
129     // Deshabilitamos interrupciones anidadas
130     INTCON1bits.NSTDIS = 1;
131 }
132
133 /*****************************************************************************
134  * Funcion:         void initAdc1(void);
135  *
136  * Resumen:         Funcion que inicializa el ADC 1
137  *
138  * Descripcion:     Funcion que configura los registros de control del ADC 1
```

```
139  *
140  * Parametros:      Ninguno
141  *
142  * Return:          Nada
143  *
144  * Notas:           Ninguna
145  ************************************************************************/
146
147 void initAdc1(void)
148 {
149     // Configuramos los registros del ADC
150     AD1CON1 = 0x00ED;   // Modo 10 bits
151                         // Formato de entero
152                         // Auto-sample y Auto-convert
153     AD1CON2 = 0x0000;   // Vrefh = Avdd; Vrefl = AVSS
154                         // Convierte CH0
155                         // Interrupcion tras cada muestreo/conversion
156     AD1CON3 = 0x0003;   // Reloj del sistema = Reloj de conversion
157                         // Tiempo auto sample = 0*Tad
158                         // Conversion a 1MHz (1us) = 12*Tad
159                         // Tad = (1/Fcy)*(ADCS+1) = (1/60M)*5 = 83.3ns (12MHz)
160     AD1CON4 = 0x0000;   // DMA no usado
161     AD1CHS123 = 0x0000; // No usamos los canales 1, 2, 3
162     AD1CHS0 = 0x0000;   // CH0+ = AN0, CH0- = Vrefl
163     AD1CSSH = 0x0000;   // No se usa
164     AD1CSSL = 0x0000;   // No se usa
165
166     IFS0bits.AD1IF = 0; // Borramos el flag de notificacion de interrupcion
167     IEC0bits.AD1IE = 1; // Habilitamos las interrupciones del ADC1
168 }
169
170 /************************************************************************
171  * Funcion:         void initAdc2(void);
172  *
173  * Resumen:         Funcion que inicializa el ADC 2
174  *
175  * Descripcion:     Funcion que configura los registros de control del ADC 2
176  *
177  * Parametros:      Ninguno
178  *
179  * Return:          Nada
180  *
181  * Notas:           Ninguna
182  ************************************************************************/
183
184 void initAdc2(void)
```

```
185 {
186     // Configuramos los registros del ADC
187     AD2CON1 = 0x00ED;   // Modo 10 bits
188                         // Formato de entero
189                         // Auto-sample y Auto-convert
190     AD2CON2 = 0x0000;   // Vrefh = Avdd; Vrefl = AVSS
191                         // Convierte CH0
192                         // Interrupcion tras cada muestreo/conversion
193     AD2CON3 = 0x0003;   // Reloj del sistema = Reloj de conversion
194                         // Tiempo auto sample = 0*Tad
195                         // Conversion a 1MHz (1us) = 12*Tad
196                         // Tad = (1/Fcy)*(ADCS+1) = (1/60M)*5 = 83.3ns (12MHz)
197     AD2CON4 = 0x0000;   // DMA no usado
198     AD2CHS123 = 0x0000; // No usamos los canales 1, 2, 3
199     AD2CHS0 = 0x0001;   // CH0+ = AN1, CH0- = Vrefl
200     // Notese que no existe AD2CSSH
201     AD2CSSL = 0x0000;   // No se usa
202
203     IFS1bits.AD2IF = 0; // Borramos el flag de notificacion de interrupcion
204     IEC1bits.AD2IE = 1; // Habilitamos las interrupciones del ADC2
205 }
206
207 /*********************************************************************
208  * Funcion:         void __attribute__((interrupt,auto_psv)) _USB1Interrupt();
209  *
210  * Resumen:         Rutina de interrupcion para el protocolo USB
211  *
212  * Descripcion:     Es la rutina de servicio de interrupcion (ISR) del protocolo
213  *                  USB, la cual se ejecuta de forma "oculta" a nuestro ciclo
214  *                  de trabajo.
215  *
216  * Parametros:      Ninguno
217  *
218  * Return:          Nada
219  *
220  * Notas:           Nada
221  ********************************************************************/
222
223 void __attribute__((interrupt,auto_psv)) _USB1Interrupt()
224 {
225     USB_HostInterruptHandler();
226 }
227
228 /*********************************************************************
229  Fin de archivo
230 */
```

```
 1 /*
 2  * Proyecto:    OscAndroid_Firmware
 3  * Archivo:     system.h
 4  * Autor:       David Ortiz de Latierro Delgado
 5  * Rev:         1.0
 6  */
 7
 8 #ifndef SYSTEM_H
 9 #define SYSTEM_H
10
11 // ****************************************************************************
12 // ****************************************************************************
13 // Seccion: Archivos #include
14 // ****************************************************************************
15 // ****************************************************************************
16
17 #include <xc.h>
18 #include <stdbool.h>
19
20 // ****************************************************************************
21 // ****************************************************************************
22 // Seccion: Declaraciones de las funciones
23 // ****************************************************************************
24 // ****************************************************************************
25
26 void init(void);
27 void initAdc1(void);
28 void initAdc2(void);
29
30
31 #endif //SYSTEM_H
32
33 /*****************************************************************************
34  Fin de archivo
35 */
```

```
 1 /*
 2  * Proyecto:    OscAndroid_Firmware
 3  * Archivo:     usb_config.c
 4  * Rev:         1.0
 5  */
 6
 7 // ****************************************************************************
 8 // ****************************************************************************
 9 // Seccion: Archivos #include
10 // ****************************************************************************
11 // ****************************************************************************
12
13 #include <usb.h>
14 #include <usb_host_android.h>
15
16 // ****************************************************************************
17 // Client Driver Function Pointer Table for the USB Embedded Host foundation
18 // ****************************************************************************
19
20 CLIENT_DRIVER_TABLE usbClientDrvTable[NUM_CLIENT_DRIVER_ENTRIES] =
21 {
22     {
23         AndroidAppInitialize,
24         AndroidAppEventHandler,
25         AndroidAppDataEventHandler,
26         0
27     },
28     {
29         AndroidAppInitialize,
30         AndroidAppEventHandler,
31         AndroidAppDataEventHandler,
32         ANDROID_INIT_FLAG_BYPASS_PROTOCOL
33     }
34 };
35
36 // ****************************************************************************
37 // USB Embedded Host Targeted Peripheral List (TPL)
38 // ****************************************************************************
39 USB_TPL usbTPL[NUM_TPL_ENTRIES] =
40 {
41       /*[1] Device identification information
42         [2] Initial USB configuration to use
43         [3] Client driver table entry
44         [4] Flags
45     ------------------------------------------------------------------
46                 [1]                       [2][3] [4]
```

```
47      ------------------------------------------------------------------*/
48      { INIT_VID_PID( 0x18D1ul, 0x2D00ul ), 0, 1, {0} }, // Android accessory
49      { INIT_VID_PID( 0x18D1ul, 0x2D01ul ), 0, 1, {0} }, // Android accessory
50      { INIT_VID_PID( 0xFFFFul, 0xFFFFul ), 0, 0, {0} }, // Enumerates everything
51  };
52
53  /********************************************************************************
54  Software License Agreement
55
56  Copyright (c) 2007-2008 Microchip Technology Inc. and its licensors.  All
57  rights reserved.
58
59  Microchip licenses to you the right to: (1) install Software on a single
60  computer and use the Software with Microchip 16-bit microcontrollers and
61  16-bit digital signal controllers ("Microchip Product"); and (2) at your
62  own discretion and risk, use, modify, copy and distribute the device
63  driver files of the Software that are provided to you in Source Code;
64  provided that such Device Drivers are only used with Microchip Products
65  and that no open source or free software is incorporated into the Device
66  Drivers without Microchip's prior written consent in each instance.
67
68  You should refer to the license agreement accompanying this Software for
69  additional information regarding your rights and obligations.
70
71  SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY
72  KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY
73  WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A
74  PARTICULAR PURPOSE. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE
75  LIABLE OR OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY,
76  CONTRIBUTION, BREACH OF WARRANTY, OR OTHER LEGAL EQUITABLE THEORY ANY
77  DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO ANY
78  INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR
79  LOST DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY,
80  SERVICES, ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY
81  DEFENSE THEREOF), OR OTHER SIMILAR COSTS.
82  ********************************************************************************/
83
84  /********************************************************************************
85   Fin de archivo
86  */
```

```
1 /*
2  * Proyecto:    OscAndroid_Firmware
3  * Archivo:     usb_config.h
4  * Rev:         1.0
5  */
6
7 // ****************************************************************************
8 // ****************************************************************************
9 // Seccion: Definicion de macros
10 // ****************************************************************************
11 // ****************************************************************************
12
13 #define _USB_CONFIG_VERSION_MAJOR 0
14 #define _USB_CONFIG_VERSION_MINOR 0
15 #define _USB_CONFIG_VERSION_DOT   12
16 #define _USB_CONFIG_VERSION_BUILD 0
17
18 #define USB_SUPPORT_HOST
19
20 #define USB_PING_PONG_MODE  USB_PING_PONG__FULL_PING_PONG
21
22 #define NUM_TPL_ENTRIES 3
23 #define NUM_CLIENT_DRIVER_ENTRIES 2
24
25 #define USB_ENABLE_TRANSFER_EVENT
26
27 #define USB_HOST_APP_DATA_EVENT_HANDLER USB_ApplicationDataEventHandler
28 //#define USB_ENABLE_SOF_EVENT
29 #define USB_ENABLE_1MS_EVENT
30
31 #define ANDROID_DEVICE_ATTACH_TIMEOUT 3000
32
33 #define USB_MAX_GENERIC_DEVICES 1
34 #define USB_NUM_CONTROL_NAKS 20
35 #define USB_SUPPORT_INTERRUPT_TRANSFERS
36 #define USB_SUPPORT_BULK_TRANSFERS
37 #define USB_SUPPORT_ISOCHRONOUS_TRANSFERS
38 #define USB_NUM_INTERRUPT_NAKS 3
39 #define USB_INITIAL_VBUS_CURRENT (100/2)
40 #define USB_INSERT_TIME (250+1)
41 #define USB_HOST_APP_EVENT_HANDLER USB_ApplicationEventHandler
42
43 #define USBTasks()                  \
44     {                               \
45         USBHostTasks();             \
46         AndroidTasks();          \
```

```
47      }
48
49  #define USBInitialize(x)            \
50      {                               \
51          USBHostInit(x);             \
52      }
53
54  /********************************************************************************
55  Software License Agreement
56
57  The software supplied herewith by Microchip Technology Incorporated
58  (the "Company") for its PICmicro(R) Microcontroller is intended and
59  supplied to you, the Company□s customer, for use solely and
60  exclusively on Microchip PICmicro Microcontroller products. The
61  software is owned by the Company and/or its supplier, and is
62  protected under applicable copyright laws. All rights are reserved.
63  Any use in violation of the foregoing restrictions may subject the
64  user to criminal sanctions under applicable laws, as well as to
65  civil liability for the breach of the terms and conditions of this
66  license.
67
68  THIS SOFTWARE IS PROVIDED IN AN □AS IS□ CONDITION. NO WARRANTIES,
69  WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT NOT LIMITED
70  TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
71  PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. THE COMPANY SHALL NOT,
72  IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL OR
73  CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
74  ********************************************************************************/
75
76  /********************************************************************************
77   Fin de archivo
78  */
```

## *F.* Código fuente de la aplicación *Android*

En este caso, el anexo incluye el código, en lenguaje *Java* y *XML*, de los archivos correspondientes a la parte lógica y gráfica de la aplicación. Se incluye, además, el archivo *manifest*, ya que también es necesario modificarlo para el correcto funcionamiento de la aplicación.

Archivo: AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.oscandroid">

    <uses-feature android:name="android.hardware.usb.accessory" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".CreditosActivity"
            android:screenOrientation="portrait"/>
        <activity
            android:name=".FullscreenActivity"
            android:configChanges="orientation|keyboardHidden|screenSize"
            android:label="@string/title_activity_fullscreen"
            android:screenOrientation="landscape"
            android:theme="@style/FullscreenTheme" />
        <activity
            android:name=".MainActivity"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED" />
            </intent-filter>

            <meta-data
                android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED"
                android:resource="@xml/accessory_filter" />
        </activity>
    </application>

</manifest>
```

Archivo: MainActivity.java

```
1 /*
2  * Proyecto:    OscAndroid_Software
3  * Archivo:     MainActivity.java
4  * Autor:       David Ortiz de Latierro Delgado
5  * Rev:         1.0
6  */
7
8 package com.example.oscandroid;
9
10 import android.content.Intent;
11 import android.support.v7.app.AppCompatActivity;
12 import android.os.Bundle;
13 import android.view.View;
14
15 public class MainActivity extends AppCompatActivity {
16
17     @Override
18     public void onWindowFocusChanged(boolean hasFocus)
19     {
20         super.onWindowFocusChanged(hasFocus);
21         if (hasFocus) {
22             hideSystemUI();
23         }
24     }
25
26     // Ocultamos la barra de acción y los botones de navegación para tener Full Screen
27     private void hideSystemUI()
28     {
29         // Habilitamos el modo "IMMERSIVE_STICKY"
30         View decorView = getWindow().getDecorView();
31         decorView.setSystemUiVisibility(
32                 View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY
33                         | View.SYSTEM_UI_FLAG_LAYOUT_STABLE
34                         | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION
35                         | View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
36                         | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION
37                         | View.SYSTEM_UI_FLAG_FULLSCREEN);
38     }
39
40     @Override
41     protected void onCreate(Bundle savedInstanceState) {
42         super.onCreate(savedInstanceState);
43         setContentView(R.layout.activity_main);
44         hideSystemUI();
45     }
46
47     // Método para el botón "Iniciar"
48     public void Inicio(View view)
49     {
50         Intent inicio = new Intent(this, FullscreenActivity.class);
51         startActivity(inicio);  // Cambiamos de activity
52     }
53
54     // Método para el botón "Créditos"
55     public void Creditos(View view)
56     {
57         Intent creditos = new Intent(this, CreditosActivity.class);
58         startActivity(creditos);    // Cambiamos de activity
59     }
60 }
61
62 //*********************************************************************************************
63 // Fin de archivo
64 //*********************************************************************************************
```

Archivo: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#2F2F31"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/tv_inicio"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/imageView"
        android:layout_centerInParent="true"
        android:layout_marginTop="16dp"
        android:text="@string/tvInicio"
        android:textColor="#FFFFFF"
        android:textSize="36sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageView" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/tv_inicio"
        android:layout_centerInParent="true"
        android:layout_marginTop="36dp"
        android:onClick="Inicio"
        android:text="@string/BotonIniciar"
        android:textAllCaps="false"
        android:textSize="24sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/tv_inicio" />

    <TextView
        android:id="@+id/tv_nombre"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/tv_version"
        android:layout_alignParentStart="true"
        android:layout_alignParentLeft="true"
        android:layout_marginStart="17dp"
        android:layout_marginLeft="17dp"
        android:layout_marginBottom="11dp"
        android:text="@string/Nombre"
        android:textColor="#FFFFFF"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <TextView
        android:id="@+id/tv_carrera"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/tv_mencion"
        android:layout_alignParentStart="true"
        android:layout_alignParentLeft="true"
        android:layout_marginStart="16dp"
        android:layout_marginLeft="16dp"
        android:layout_marginBottom="8dp"
        android:text="@string/Carrera"
```

Archivo: activity_main.xml

```
        android:textColor="#FFFFFF"
        android:textStyle="bold"
        app:layout_constraintBottom_toTopOf="@+id/tv_mencion"
        app:layout_constraintStart_toStartOf="parent" />

    <TextView
        android:id="@+id/tv_mencion"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/tv_universidad"
        android:layout_alignParentStart="true"
        android:layout_alignParentLeft="true"
        android:layout_marginStart="16dp"
        android:layout_marginLeft="16dp"
        android:layout_marginBottom="12dp"
        android:text="@string/Mencion"
        android:textColor="#FFFFFF"
        android:textStyle="bold"
        app:layout_constraintBottom_toTopOf="@+id/tv_universidad"
        app:layout_constraintStart_toStartOf="parent" />

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentStart="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:layout_centerInParent="true"
        android:layout_centerHorizontal="true"
        android:layout_marginStart="127dp"
        android:layout_marginLeft="127dp"
        android:layout_marginTop="64dp"
        android:layout_marginEnd="126dp"
        android:layout_marginRight="126dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.497"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:srcCompat="@drawable/inicio" />

    <Button
        android:id="@+id/button_creditos"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentBottom="true"
        android:layout_marginEnd="16dp"
        android:layout_marginRight="16dp"
        android:layout_marginBottom="17dp"
        android:background="#2F2F31"
        android:onClick="Creditos"
        android:text="@string/Creditos"
        android:textColor="#FFFFFF"
        android:textSize="14sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />

    <TextView
        android:id="@+id/tv_universidad"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

Archivo: activity_main.xml

```
131         android:layout_above="@+id/tv_nombre"
132         android:layout_alignParentStart="true"
133         android:layout_alignParentLeft="true"
134         android:layout_marginStart="15dp"
135         android:layout_marginLeft="15dp"
136         android:layout_marginBottom="12dp"
137         android:text="@string/Universidad"
138         android:textColor="#FFFFFFFF"
139         android:textStyle="bold"
140         app:layout_constraintBottom_toTopOf="@+id/tv_nombre"
141         app:layout_constraintStart_toStartOf="parent" />
142
143     <TextView
144         android:id="@+id/tv_version"
145         android:layout_width="wrap_content"
146         android:layout_height="wrap_content"
147         android:layout_alignParentStart="true"
148         android:layout_alignParentLeft="true"
149         android:layout_alignParentBottom="true"
150         android:layout_marginStart="16dp"
151         android:layout_marginLeft="16dp"
152         android:layout_marginBottom="24dp"
153         android:text="@string/Version"
154         android:textColor="#FFFFFF"
155         android:textStyle="bold" />
156 </RelativeLayout>
```

Archivo: CreditosActivity.java

```
/*
 * Proyecto:    OscAndroid_Software
 * Archivo:     CreditosActivity.java
 * Autor:       David Ortiz de Latierro Delgado
 * Rev:         1.0
 */

package com.example.oscandroid;

import android.content.Intent;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ImageButton;
import android.widget.ImageView;
import android.widget.TextView;

public class CreditosActivity extends AppCompatActivity implements OnClickListener {

    // Definimos los objetos del layout
    private ImageView imagen1, imagen2;
    private TextView enlace;
    private ImageButton volver;

    // Cadena para guardar las URL de las paginas web
    private String direccion;

    @Override
    public void onWindowFocusChanged(boolean hasFocus)
    {
        super.onWindowFocusChanged(hasFocus);
        if (hasFocus) {
            hideSystemUI();
        }
    }

    // Ocultamos la barra de acción y los botones de navegación para tener Full Screen
    private void hideSystemUI()
    {
        // Habilitamos el modo "IMMERSIVE_STICKY"
        View decorView = getWindow().getDecorView();
        decorView.setSystemUiVisibility(
                View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY
                        | View.SYSTEM_UI_FLAG_LAYOUT_STABLE
                        | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION
                        | View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
                        | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION
                        | View.SYSTEM_UI_FLAG_FULLSCREEN);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_creditos);
        hideSystemUI();

        // Relaciones entre parte logica y parte gráfica
        imagen1 = (ImageView)findViewById(R.id.imagen1);
        imagen2 = (ImageView)findViewById(R.id.imagen2);
        enlace = (TextView)findViewById(R.id.enlace);

        // Inicializamos direccion para evitar errores
        direccion = "";
```

Archivo: CreditosActivity.java

```
66
67          // Definimos los setOnClick
68          imagen1.setOnClickListener(this);
69          imagen2.setOnClickListener(this);
70          enlace.setOnClickListener(this);
71      }
72
73      @Override
74      public void onClick(View view)
75      {
76          // Comprobamos sobre cual se ha pulsado
77          switch(view.getId())
78          {
79              case R.id.imagen1:
80                  direccion = "http://www.uva.es/export/sites/uva/";
81                  abrirWeb(direccion);
82                  break;
83              case R.id.imagen2:
84                  direccion = "https://www.tel.uva.es/";
85                  abrirWeb(direccion);
86                  break;
87              case R.id.enlace:
88                  direccion = "url_3";
89                  abrirWeb(direccion);
90                  break;
91              default:
92                  break;
93          }
94      }
95
96      // Función para lanzar el navegador configurado por defecto con la URL seleccionada
97      public void abrirWeb(String d)
98      {
99          Uri uri = Uri.parse(d);
100         Intent intentNav = new Intent(Intent.ACTION_VIEW, uri);
101         startActivity(intentNav);
102     }
103
104     // Método para el botón "Volver"
105     public void volver(View view)
106     {
107         Intent volver = new Intent(this, MainActivity.class);
108         startActivity(volver);    // Cambiamos de activity
109     }
110 }
111
112 //*************************************************************************************************
113 // Fin de archivo
114 //*************************************************************************************************
```

Archivo: activity_creditos.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#2F2F31"
    tools:context=".CreditosActivity">

    <ScrollView
        android:layout_width="395dp"
        android:layout_height="715dp"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginBottom="8dp"
        android:background="#00FFFFFF"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="#002F2F31"
            android:orientation="vertical" >

            <ImageButton
                android:id="@+id/boton_volver"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="end"
                android:layout_marginTop="16dp"
                android:layout_marginRight="16dp"
                android:background="#00FFFFFF"
                android:onClick="volver"
                app:srcCompat="@android:drawable/ic_menu_revert" />

            <TextView
                android:id="@+id/volver"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginRight="14dp"
                android:gravity="right"
                android:text="@string/volver"
                android:textColor="#FFFFFF"
                android:textStyle="bold" />

            <TextView
                android:id="@+id/Titulo"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="24dp"
                android:gravity="center"
                android:text="@string/TFG"
                android:textColor="#FFFFFF"
                android:textSize="24sp" />

            <TextView
                android:id="@+id/trabajo"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_marginTop="64dp"
```

Archivo: activity_creditos.xml

```
66                  android:gravity="center"
67                  android:text="@string/trabajo"
68                  android:textColor="#FFFFFF"
69                  android:textSize="18sp" />
70
71              <TextView
72                  android:id="@+id/titulacion"
73                  android:layout_width="match_parent"
74                  android:layout_height="wrap_content"
75                  android:layout_marginTop="12dp"
76                  android:gravity="center"
77                  android:text="@string/titulacion"
78                  android:textColor="#FFFFFF"
79                  android:textSize="18sp" />
80
81              <TextView
82                  android:id="@+id/especialidad"
83                  android:layout_width="match_parent"
84                  android:layout_height="wrap_content"
85                  android:layout_marginTop="12dp"
86                  android:gravity="center"
87                  android:text="@string/especialidad"
88                  android:textColor="#FFFFFF"
89                  android:textSize="18sp" />
90
91              <TextView
92                  android:id="@+id/autor"
93                  android:layout_width="match_parent"
94                  android:layout_height="wrap_content"
95                  android:layout_marginTop="64dp"
96                  android:gravity="center"
97                  android:text="@string/autor"
98                  android:textColor="#FFFFFF"
99                  android:textSize="18sp" />
100
101             <TextView
102                 android:id="@+id/yo"
103                 android:layout_width="match_parent"
104                 android:layout_height="wrap_content"
105                 android:gravity="center"
106                 android:text="@string/Nombre"
107                 android:textColor="#FFFFFF"
108                 android:textSize="18sp"
109                 android:textStyle="bold" />
110
111             <TextView
112                 android:id="@+id/tutores"
113                 android:layout_width="match_parent"
114                 android:layout_height="wrap_content"
115                 android:layout_marginTop="12dp"
116                 android:gravity="center"
117                 android:text="@string/tutores"
118                 android:textColor="#FFFFFF"
119                 android:textSize="18sp" />
120
121             <TextView
122                 android:id="@+id/ivan"
123                 android:layout_width="match_parent"
124                 android:layout_height="wrap_content"
125                 android:gravity="center"
126                 android:text="@string/ivan"
127                 android:textColor="#FFFFFF"
128                 android:textSize="18sp"
129                 android:textStyle="bold" />
130
```

Archivo: activity_creditos.xml

```
131             <TextView
132                 android:id="@+id/miguel"
133                 android:layout_width="match_parent"
134                 android:layout_height="wrap_content"
135                 android:gravity="center"
136                 android:text="@string/miguel"
137                 android:textColor="#FFFFFF"
138                 android:textSize="18sp"
139                 android:textStyle="bold" />
140 
141             <TextView
142                 android:id="@+id/enlace"
143                 android:layout_width="match_parent"
144                 android:layout_height="wrap_content"
145                 android:layout_marginTop="24dp"
146                 android:gravity="center_horizontal"
147                 android:text="TextView"
148                 android:textColor="#FFFFFF" />
149 
150             <ImageView
151                 android:id="@+id/imagen2"
152                 android:layout_width="match_parent"
153                 android:layout_height="wrap_content"
154                 android:layout_marginTop="64dp"
155                 app:srcCompat="@drawable/etsit" />
156 
157             <ImageView
158                 android:id="@+id/imagen1"
159                 android:layout_width="match_parent"
160                 android:layout_height="wrap_content"
161                 android:layout_marginTop="64dp"
162                 android:layout_marginBottom="24dp"
163                 app:srcCompat="@drawable/logouva" />
164 
165             <TextView
166                 android:id="@+id/version_creditos"
167                 android:layout_width="match_parent"
168                 android:layout_height="wrap_content"
169                 android:layout_marginTop="24dp"
170                 android:layout_marginBottom="24dp"
171                 android:text="@string/Version"
172                 android:textColor="#FFFFFF"
173                 android:textStyle="bold" />
174         </LinearLayout>
175     </ScrollView>
176 </RelativeLayout>
```

Archivo: FullScreenActivity.java

```
 1 /*
 2  * Proyecto:    OscAndroid_Software
 3  * Archivo:     FullscreenActivity.java
 4  * Autor:       David Ortiz de Latierro Delgado
 5  * Rev:         1.0
 6  */
 7
 8 package com.example.oscandroid;
 9
10 import android.app.Activity;
11 import android.app.Dialog;
12 import android.app.PendingIntent;
13 import android.content.BroadcastReceiver;
14 import android.content.Context;
15 import android.content.Intent;
16 import android.content.IntentFilter;
17 import android.graphics.Bitmap;
18 import android.hardware.usb.UsbAccessory;
19 import android.hardware.usb.UsbManager;
20 import android.media.MediaScannerConnection;
21 import android.net.Uri;
22 import android.os.Environment;
23 import android.os.ParcelFileDescriptor;
24 import android.support.v7.app.AppCompatActivity;
25 import android.os.Bundle;
26 import android.util.Log;
27 import android.view.View;
28 import android.view.ViewTreeObserver;
29 import android.view.Window;
30 import android.widget.AdapterView;
31 import android.widget.ArrayAdapter;
32 import android.widget.CheckBox;
33 import android.widget.Spinner;
34 import android.widget.TextView;
35 import android.widget.Toast;
36 import android.widget.ToggleButton;
37
38 import java.io.File;
39 import java.io.FileDescriptor;
40 import java.io.FileInputStream;
41 import java.io.FileOutputStream;
42 import java.io.IOException;
43 import java.io.OutputStreamWriter;
44 import java.text.DateFormat;
45 import java.text.DecimalFormat;
46 import java.text.SimpleDateFormat;
47 import java.util.Arrays;
48 import java.util.Date;
49
50 public class FullscreenActivity extends AppCompatActivity implements Runnable {
51
52     // Definimos las constantes necesarias para el control del protocolo USB en modo 'Accessory'
53     private static final String TAG = "FullScreen";
54     private static final String ACTION_USB_PERMISSION =
55             "com.example.oscandroid.action.USB_PERMISSION";
56
57     private UsbManager mUsbManager;
58     private PendingIntent mPermissionIntent;
59     private boolean mPermissionRequestPending;
60
61     UsbAccessory mAccessory;
62     ParcelFileDescriptor mFileDescriptor;
63     FileInputStream mInputStream;
64     FileOutputStream mOutputStream;
65
```

Archivo: FullScreenActivity.java

```
66      // Definimos los objetos del layout
67      private Spinner spinner_vert1, spinner_vert2, spinner_tiempos, spinner_trigger;
68      private TextView tv_amp1_val, tv_amp2_val, tv_frec1_val, tv_frec2_val;
69      private ToggleButton boton_ch1, boton_ch2;
70      private Lienzo lienzo;
71
72      // Variables para envios
73      byte status = 0x00;             // Byte de estado de la interfaz grafica
74      byte[] comando = new byte[3];   // Array de bytes a modo de buffer de salida
75
76      // Variables para guardar medidas y valores
77      float xmax;
78      float ymax;
79      private boolean ymax_valido = false;
80
81      // Variables para el escalado vertical
82      float [] kes = {0,0,0};
83      float constante1 = kes[0];
84      float constante2 = kes[0];
85      int escala_voltajes1 = -1;
86      int escala_voltajes2 = -1;
87
88      // Variables para calcular la amplitud real de las señales de entrada
89      double [] factorAmp = {10,20,50,0.1,0.2,0.5,1,2,5,10,20,50};
90      String [] coletillaAmp = {" mV"," mV"," mV"," V"," V"," V"," V"," V"," V"," V"," V"," V"};
91
92      // Variables para el escalado horizontal
93      int [] tasa_avance = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
94      int [] num_muestras = {68,136,340,680};
95      int escala_tiempos = 0;
96
97      // Variables para calcular la frecuencia real de las señales de entrada
98      double [] factorFrec = {0.1,0.2,0.5,1,2,5,0.01,0.02,0.05,0.1,0.2,0.5,1,2,5};
99      String [] coletillaFrec = {" KHz"," KHz"," KHz"," KHz"," KHz"," KHz"," Hz"," Hz"," Hz",
100                                " Hz"," Hz"," Hz"," Hz"," Hz"," Hz"};
101
102     // Variable auxiliar para bucles
103     int aux = 0;
104
105     // Variables para el calculo de amplitudes y frecuencias
106     int valor11, valor21, valor12, valor22, frec11, frec21, frec12, frec22;
107     int yalto1 = 0;
108     int yalto2 = 0;
109     int ybajo1 = 0;
110     int ybajo2 = 0;
111
112     // Variables para guardar los puntos de la representacion grafica
113     int xbefore = 0;
114     int xactual = 0;
115     float ybefore1 = 0;
116     float ybefore2 = 0;
117     float yactual1 = 0;
118     float yactual2 = 0;
119
120     int dato1 = 0;
121     int dato2 = 0;
122
123     // Variable para almacenar los datos leídos en caso de querer guardar una copia
124     int [] copiaDatos;
125     int [] intermedio;
126
127     private boolean conversiones = false;
128     public static boolean pintado = false;
129     private boolean fin_pantalla = false;
130     String numberAsString = "";
```

Archivo: FullScreenActivity.java

```
131     String numberAsString2 = "";
132     String numberAsString3 = "";
133     String numberAsString4 = "";
134     public boolean sdDisponible = false;
135     public boolean sdAccesoEscritura = false;
136
137     int activoCanal1 = 0;
138     int activoCanal2 = 0;
139     int triggerPosicion = -1;
140     boolean frecuenciaConseguida1 = false;
141     boolean frecuenciaConseguida2 = false;
142     boolean maxEncontrado = false;
143     boolean minEncontrado = false;
144     boolean maxEncontrado2 = false;
145     boolean minEncontrado2 = false;
146
147     // ********************************************************************************************
148
149     @Override
150     public void onWindowFocusChanged(boolean hasFocus)
151     {
152         super.onWindowFocusChanged(hasFocus);
153         if (hasFocus)
154         {
155             hideSystemUI();
156         }
157     }
158
159     // Ocultamos la barra de acción y los botones de navegación para tener Full Screen
160     private void hideSystemUI()
161     {
162         // Habilitamos el modo "IMMERSIVE_STICKY"
163         View decorView = getWindow().getDecorView();
164         decorView.setSystemUiVisibility(
165                 View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY
166                         | View.SYSTEM_UI_FLAG_LAYOUT_STABLE
167                         | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION
168                         | View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
169                         | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION
170                         | View.SYSTEM_UI_FLAG_FULLSCREEN);
171     }
172
173     // ********************************************************************************************
174
175     // Obtenemos permiso para comunicarnos con el accesorio por USB
176     private final BroadcastReceiver mUsbReceiver = new BroadcastReceiver()
177     {
178         @Override
179         public void onReceive(Context context, Intent intent)
180         {
181             String action = intent.getAction();
182             if(ACTION_USB_PERMISSION.equals(action))
183             {
184                 synchronized (this)
185                 {
186                     UsbAccessory accessory =
187                             (UsbAccessory) intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
188                     // Si tenemos permiso, nos conectamos al accesorio
189                     if(intent.getBooleanExtra(UsbManager.EXTRA_PERMISSION_GRANTED,
190                             false))
191                     {
192                         if(accessory != null) {
193                             openAccessory(accessory);   // Abrimos la comunicación
194                         }
195                     }
```

Archivo: FullScreenActivity.java

```
196                     else
197                     {
198                         Log.d(TAG, "Permiso denegado para el accesorio" + accessory);
199                     }
200                     mPermissionRequestPending = false;
201                 }
202             }   // Si desconectamos el accesorio
203             else if(UsbManager.ACTION_USB_ACCESSORY_DETACHED.equals(action))
204             {
205                 UsbAccessory accessory =
206                         (UsbAccessory) intent.getParcelableExtra(UsbManager.EXTRA_ACCESSORY);
207                 if(accessory != null && accessory.equals(mAccessory))
208                 {
209                     closeAccessory();   // Cerramos la comunicación
210                 }
211             }
212         }
213     };
214
215     // ********************************************************************************************
216
217     @Override
218     protected void onCreate(Bundle savedInstanceState) {
219         super.onCreate(savedInstanceState);
220
221         mUsbManager = (UsbManager) getSystemService(Context.USB_SERVICE);
222         mPermissionIntent = PendingIntent.getBroadcast(this, 0,
223                 new Intent(ACTION_USB_PERMISSION), 0);
224         IntentFilter filter = new IntentFilter(ACTION_USB_PERMISSION);
225         filter.addAction(UsbManager.ACTION_USB_ACCESSORY_DETACHED);
226         registerReceiver(mUsbReceiver, filter);
227
228         if(getLastNonConfigurationInstance() != null)
229         {
230             mAccessory = (UsbAccessory) getLastNonConfigurationInstance();
231             openAccessory(mAccessory);
232         }
233
234         setContentView(R.layout.activity_fullscreen);
235
236         // Relaciones entre parte lógica y gráfica
237         tv_amp1_val = (TextView)findViewById(R.id.tv_amp1_val);
238         tv_amp2_val = (TextView)findViewById(R.id.tv_amp2_val);
239         tv_frec1_val = (TextView)findViewById(R.id.tv_frec1_val);
240         tv_frec2_val = (TextView)findViewById(R.id.tv_frec2_val);
241
242         spinner_vert1 = (Spinner)findViewById(R.id.spinner_vert1);
243         spinner_vert2 = (Spinner)findViewById(R.id.spinner_vert2);
244         spinner_tiempos = (Spinner)findViewById(R.id.spinner_tiempos);
245         spinner_trigger = (Spinner)findViewById(R.id.spinner_trigger);
246
247         boton_ch1 = (ToggleButton)findViewById(R.id.boton_ch1);
248         boton_ch2 = (ToggleButton)findViewById(R.id.boton_ch2);
249
250         lienzo = (Lienzo)findViewById(R.id.lienzo);
251
252         // ********************************************************************************************
253
254         // Vectores para opciones de los spinners
255         String [] voltajes = {"10mV/DIV","20mV/DIV","50mV/DIV","0.1V/DIV","0.2V/DIV","0.5V/DIV",
256                               "1V/DIV","2V/DIV","5V/DIV","10V/DIV","20V/DIV","50V/DIV"};
257         String [] tiempos = {"10us/DIV","20us/DIV","50us/DIV","0.1ms/DIV","0.2ms/DIV","0.5ms/DIV",
258                              "1ms/DIV","2ms/DIV","5ms/DIV","10ms/DIV","20ms/DIV","50ms/DIV",
259                              "100ms/DIV","200ms/DIV","500ms/DIV"};
260         String [] triggers = {"Fl. subida (1)","Fl. subida (2)","Fl. bajada (1)","Fl. bajada (2)"};
```

Archivo: FullScreenActivity.java

```
261
262         // Comunicación entre los vectores y los spinners
263         ArrayAdapter <String> adapter_voltajes1 = new ArrayAdapter<String>(this,
264                 R.layout.spinner_item_vert1, voltajes);
265         ArrayAdapter <String> adapter_voltajes2 = new ArrayAdapter<String>(this,
266                 R.layout.spinner_item_vert2, voltajes);
267         ArrayAdapter <String> adapter_tiempos = new ArrayAdapter<String>(this,
268                 R.layout.spinner_item_custom, tiempos);
269         ArrayAdapter <String> adapter_triggers = new ArrayAdapter<String>(this,
270                 R.layout.spinner_item_custom, triggers);
271
272         // Colocamos los ArrayAdapter dentro de los spinners
273         spinner_vert1.setAdapter(adapter_voltajes1);
274         spinner_vert2.setAdapter(adapter_voltajes2);
275         spinner_tiempos.setAdapter(adapter_tiempos);
276         spinner_trigger.setAdapter(adapter_triggers);
277
278         // ********************************************************************************************
279
280         // Listener para los toggle button
281         boton_ch1.setOnClickListener(new View.OnClickListener()
282         {
283             @Override
284             public void onClick(View view)  // Si pulsamos el botón CH1
285             {
286                 if(boton_ch1.isChecked())
287                 {
288                     if(activoCanal2 == 1)
289                     {
290                         // Activamos el muestreo del ADC1
291                         status = (byte) (status + 1);
292                         activoCanal1 = 1;
293                     }
294                     else
295                     {
296                         if((triggerPosicion == -1)||(triggerPosicion == 0)||(triggerPosicion == 2))
297                         {
298                             // Activamos el muestreo del ADC1
299                             status = (byte) (status + 1);
300                             activoCanal1 = 1;
301                         }
302                         else
303                         {
304                             Toast.makeText(getApplicationContext(),
305                                     "Seleccione un Trigger apropiado para el canal",
306                                     Toast.LENGTH_SHORT).show();
307                             boton_ch1.setChecked(false);
308                         }
309                     }
310                 }
311                 else
312                 {
313                     // Desactivamos el muestro del ADC1
314                     status = (byte) (status - 1);
315                     activoCanal1 = 0;
316                 }
317             }
318         });
319         boton_ch2.setOnClickListener(new View.OnClickListener()
320         {
321             @Override
322             public void onClick(View view)  // Si pulsamos el botón CH2
323             {
324                 if(boton_ch2.isChecked())
325                 {
```

Archivo: FullScreenActivity.java

```
326                     if(activoCanal1 == 1)
327                     {
328                         // Activamos el muestro del ADC2
329                         status = (byte) (status + 2);
330                         activoCanal2 = 1;
331                     }
332                     else
333                     {
334                         if((triggerPosicion == 1) || (triggerPosicion == 3))
335                         {
336                             // Activamos el muestro del ADC2
337                             status = (byte) (status + 2);
338                             activoCanal2 = 1;
339                         }
340                         else
341                         {
342                             Toast.makeText(getApplicationContext(),
343                                     "Seleccione un Trigger apropiado para el canal",
344                                     Toast.LENGTH_SHORT).show();
345                             boton_ch2.setChecked(false);
346                         }
347                     }
348                 }
349                 else
350                 {
351                     // Deactivamos el muestro del ADC2
352                     status = (byte) (status - 2);
353                     activoCanal2 = 0;
354                 }
355             }
356         });
357 
358         // ********************************************************************************************
359 
360         // Listener para los spinners
361         spinner_tiempos.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener()
362         {
363             @Override
364             public void onItemSelected(AdapterView<?> parent, View view, int position, long id)
365             {
366                 // Hacemos un switch de la posicion del vector de tiempos, para configurar el
367                 // byte de estado con la escala que estamos utilizando para pasarsela al accesorio
368                 escala_tiempos = position;
369                 switch (position)
370                 {
371                     case 0:
372                         status = (byte) (status & 0x0F);
373                         status = (byte) (status + 0);
374                         break;
375                     case 1:
376                         status = (byte) (status & 0x0F);
377                         status = (byte) (status + 16);
378                         break;
379                     case 2:
380                         status = (byte) (status & 0x0F);
381                         status = (byte) (status + 32);
382                         break;
383                     case 3:
384                         status = (byte) (status & 0x0F);
385                         status = (byte) (status + 48);
386                         break;
387                     case 4:
388                         status = (byte) (status & 0x0F);
389                         status = (byte) (status + 64);
390                         break;
```

Archivo: FullScreenActivity.java

```
391                     case 5:
392                         status = (byte) (status & 0x0F);
393                         status = (byte) (status + 80);
394                         break;
395                     case 6:
396                         status = (byte) (status & 0x0F);
397                         status = (byte) (status + 96);
398                         break;
399                     case 7:
400                         status = (byte) (status & 0x0F);
401                         status = (byte) (status + 112);
402                         break;
403                     case 8:
404                         status = (byte) (status & 0x0F);
405                         status = (byte) (status + 128);
406                         break;
407                     case 9:
408                         status = (byte) (status & 0x0F);
409                         status = (byte) (status + 144);
410                         break;
411                     case 10:
412                         status = (byte) (status & 0x0F);
413                         status = (byte) (status + 160);
414                         break;
415                     case 11:
416                         status = (byte) (status & 0x0F);
417                         status = (byte) (status + 176);
418                         break;
419                     case 12:
420                         status = (byte) (status & 0x0F);
421                         status = (byte) (status + 192);
422                         break;
423                     case 13:
424                         status = (byte) (status & 0x0F);
425                         status = (byte) (status + 208);
426                         break;
427                     case 14:
428                         status = (byte) (status & 0x0F);
429                         status = (byte) (status + 224);
430                         break;
431                     default:
432                         break;
433                 }
434             }
435 
436             @Override
437             public void onNothingSelected(AdapterView<?> parent)
438             {
439             }
440         });
441         spinner_trigger.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener()
442         {
443             @Override
444             public void onItemSelected(AdapterView<?> parent, View view, int position, long id)
445             {
446                 triggerPosicion = position;
447                 // Hacemos un switch de la posicion del vector de trigger, para configurar el
448                 // byte de estado con el trigger que queremos que utilice el accesorio USB
449                 switch (position)
450                 {
451                     case 0:
452                         status = (byte) (status & 0xF3);
453                         status = (byte) (status + 0);
454                         break;
455                     case 1:
```

Archivo: FullScreenActivity.java

```
456                         status = (byte) (status & 0xF3);
457                         status = (byte) (status + 8);
458                         break;
459                     case 2:
460                         status = (byte) (status & 0xF3);
461                         status = (byte) (status + 4);
462                         break;
463                     case 3:
464                         status = (byte) (status & 0xF3);
465                         status = (byte) (status + 12);
466                         break;
467                     default:
468                         break;
469                 }
470             }
471 
472             @Override
473             public void onNothingSelected(AdapterView<?> parent)
474             {
475             }
476         });
477         spinner_vert1.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener()
478         {
479             @Override
480             public void onItemSelected(AdapterView<?> parent, View view, int position, long id)
481             {
482                 // Hacemos un switch de la posicion del vector de voltajes del CH1, para ver qué
483                 // escala estamos utilizando y poder calcular la constante de proporcion
484                 escala_voltajes1 = position;
485                 switch (position)
486                 {
487                     case 0:
488                         constante1 = (float) kes[0];
489                         break;
490                     case 1:
491                         constante1 = (float) kes[1];
492                         break;
493                     case 2:
494                         constante1 = (float) kes[2];
495                         break;
496                     case 3:
497                         constante1 = (float) kes[0];
498                         break;
499                     case 4:
500                         constante1 = (float) kes[1];
501                         break;
502                     case 5:
503                         constante1 = (float) kes[2];
504                         break;
505                     case 6:
506                         constante1 = (float) kes[0];
507                         break;
508                     case 7:
509                         constante1 = (float) kes[1];
510                         break;
511                     case 8:
512                         constante1 = (float) kes[2];
513                         break;
514                     case 9:
515                         constante1 = (float) kes[0];
516                         break;
517                     case 10:
518                         constante1 = (float) kes[1];
519                         break;
520                     case 11:
```

Archivo: FullScreenActivity.java

```
521                         constante1 = (float) kes[2];
522                         break;
523                     default:
524                         break;
525                 }
526             }
527 
528             @Override
529             public void onNothingSelected(AdapterView<?> parent)
530             {
531             }
532         });
533         spinner_vert2.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener()
534         {
535             @Override
536             public void onItemSelected(AdapterView<?> parent, View view, int position, long id)
537             {
538                 // Hacemos un switch de la posicion del vector de voltajes del CH2, para ver qué
539                 // escala estamos utilizando y poder calcular la constante de proporcion
540                 escala_voltajes2 = position;
541                 switch (position)
542                 {
543                     case 0:
544                         constante2 = (float) kes[0];
545                         break;
546                     case 1:
547                         constante2 = (float) kes[1];
548                         break;
549                     case 2:
550                         constante2 = (float) kes[2];
551                         break;
552                     case 3:
553                         constante2 = (float) kes[0];
554                         break;
555                     case 4:
556                         constante2 = (float) kes[1];
557                         break;
558                     case 5:
559                         constante2 = (float) kes[2];
560                         break;
561                     case 6:
562                         constante2 = (float) kes[0];
563                         break;
564                     case 7:
565                         constante2 = (float) kes[1];
566                         break;
567                     case 8:
568                         constante2 = (float) kes[2];
569                         break;
570                     case 9:
571                         constante2 = (float) kes[0];
572                         break;
573                     case 10:
574                         constante2 = (float) kes[1];
575                         break;
576                     case 11:
577                         constante2 = (float) kes[2];
578                         break;
579                     default:
580                         break;
581                 }
582             }
583 
584             @Override
585             public void onNothingSelected(AdapterView<?> parent)
```

Archivo: FullScreenActivity.java

```
586             {
587             }
588         });
589 
590         //*********************************************************************************************
591 
592         // Esperamos a calcular el valor de Xmax para cuando nos lo pida el accesorio USB
593         lienzo.getViewTreeObserver().addOnGlobalLayoutListener(
594                 new ViewTreeObserver.OnGlobalLayoutListener()
595         {
596             @Override
597             public void onGlobalLayout()
598             {                       // Una vez en este punto ya podemos calcular las dimensiones del
599                                     // lienzo donde representaremos las señales de entrada
600                 lienzo.getViewTreeObserver().removeGlobalOnLayoutListener(this);
601 
602                 // Calculamos xmax
603                 xmax = lienzo.getWidth();
604 
605                 // Calculamos el vector de tasa de avance en X, una vez tenemos el valor de Xmax
606                 for(aux = 0; aux < 15; aux++)
607                 {
608                     if(aux < 4)
609                     {
610                         if(xmax > num_muestras[aux])
611                         {
612                             tasa_avance[aux] = (int) (xmax / num_muestras[aux]);
613                         }
614                         else
615                         {
616                             tasa_avance[aux] = 1;
617                         }
618                     }
619                     else
620                     {
621                         tasa_avance[aux] = 1;
622                     }
623                 }
624 
625                 intermedio = new int[(int) ((xmax*4) +3)];
626                 for(aux = 0; aux < ((xmax*4) +3); aux++)
627                 {
628                     intermedio[aux] = 0;
629                 }
630 
631                 // Notificamos al accesorio USB que ya estamos listos para funcionar
632                 comando[0] = (byte) 0xf5;
633                 enviaDato(comando,0,1);
634             }
635         });
636     }
637 
638     // ********************************************************************************************
639 
640     @Override
641     public void onResume()
642     {
643         super.onResume();
644 
645         Intent intent = getIntent();
646         if(mInputStream != null && mOutputStream != null)
647         {
648             return;
649         }
650 
```

Archivo: FullScreenActivity.java

```
651         UsbAccessory[] accessories = mUsbManager.getAccessoryList();
652         UsbAccessory accessory = (accessories == null ? null : accessories[0]);
653         if(accessory != null)
654         {
655             if(mUsbManager.hasPermission(accessory))
656             {
657                 openAccessory(accessory);
658             }
659             else
660             {
661                 synchronized(mUsbReceiver)
662                 {
663                     if(!mPermissionRequestPending)
664                     {
665                         mUsbManager.requestPermission(accessory, mPermissionIntent);
666                         mPermissionRequestPending = true;
667                     }
668                 }
669             }
670         }
671         else
672         {
673             Log.d(TAG, "mAccessory is null");
674         }
675     }
676 
677     // ************************************************************************************************
678 
679     @Override
680     public void onPause()
681     {
682         super.onPause();
683         closeAccessory();
684     }
685 
686     // ************************************************************************************************
687 
688     @Override
689     public void onDestroy()
690     {
691         unregisterReceiver(mUsbReceiver);
692         super.onDestroy();
693     }
694 
695     // ************************************************************************************************
696 
697     // Función para abrir la comunicación con el accesorio USB
698     private void openAccessory(UsbAccessory accessory)
699     {
700         mFileDescriptor = mUsbManager.openAccessory(accessory);
701         if(mFileDescriptor != null)
702         {
703             mAccessory = accessory;
704             FileDescriptor fd = mFileDescriptor.getFileDescriptor();
705             mInputStream = new FileInputStream(fd);
706             mOutputStream = new FileOutputStream(fd);
707             Thread thread = new Thread(null, this, "AccessoryThread");
708             thread.start();
709             Log.d(TAG, "Accesorio abierto" + accessory);
710         }
711         else
712         {
713             Log.d(TAG, "Fallo al abrir el accesorio");
714         }
715     }
```

Archivo: FullScreenActivity.java

```
716
717     // ********************************************************************************************
718
719     // Función para cerrar la comunicación con el accesorio USB
720     private void closeAccessory()
721     {
722         try
723         {
724             if(mFileDescriptor != null)
725             {
726                 mFileDescriptor.close();
727             }
728         }
729         catch (IOException e)
730         {
731         }
732         finally
733         {
734             mFileDescriptor = null;
735             mAccessory = null;
736         }
737     }
738
739     // ********************************************************************************************
740
741     // Funcion para pasar a entero los valores (byte) de los ADCs
742     public int composeInt(byte msb, byte lsb)
743     {
744         int valor = (int) msb & 0xFF;
745         valor *= 256;
746         valor += (int) lsb & 0xFF;
747
748         return valor;
749     }
750
751     // ********************************************************************************************
752
753     // Funcion para enviar datos al accesorio USB
754     public void enviaDato(byte dato[], int offset, int len)
755     {
756         byte[] buffer = new byte[3];     // El dato más largo que enviamos es el valor de Xmax y lo
757                                          // hacemos con tres bytes
758         buffer = dato;
759         if(mOutputStream != null)
760         {
761             try
762             {
763                 mOutputStream.write(buffer,offset,len);
764             }
765             catch(IOException e)
766             {
767                 Log.e(TAG, "Fallo de escritura");
768             }
769         }
770     }
771
772     // ********************************************************************************************
773
774     //Funcion para leer datos desde el accesorio USB (Subrutina de trabajo)
775     public void run()
776     {
777         int ret = 0;                  // Variable para guardar el num. de bytes leidos
778         byte[] buffer = new byte[16384];     // Tamaño del buffer establecido por seguridad
779         int i;
780
```

Archivo: FullScreenActivity.java

```
781         while(ret >= 0)
782         {
783             try
784             {
785                 ret = mInputStream.read(buffer);    // Guardamos los datos en el buffer y asignamos
786                 Log.d("Leo", String.valueOf(ret));
787             }                                       // a 'ret' el número de bytes leidos
788             catch(IOException e)
789             {
790                 break;
791             }
792
793             i = 0;
794             while(i < ret)                          // Mientras queden bytes que leer
795             {
796                 switch (buffer[i])                   // Comprobamos si es un dato o una orden
797                 {
798                     case (byte) 0xff:               // Si comienza con 0xFF es una 'orden'
799                         switch (buffer[i + 1])        // Comprobamos de que orden se trata
800                         {
801                             case (byte) 0xff:       // Nos solicitan el valor de Xmax
802
803                                 comando[0] = (byte) 0xFF;                          // Cabecera
804                                 comando[1] = (byte) (xmax / 256);                  // MSB xmax
805                                 comando[2] = (byte) (xmax - (comando[1] * 256));  // LSB xmax
806
807                                 enviaDato(comando, 0, 3);           // Enviamos el dato
808                                 break;
809
810                             case (byte) 0xf0:       // Nos solicitan el valor de status
811
812                                 comando[0] = status;                              // Valor de status
813
814                                 enviaDato(comando, 0, 1);           // Enviamos el dato
815                                 break;
816
817                             default:                // Orden desconocida
818                                 break;
819                         }
820                         i += 2; // Todas las ordenes que nos envian son de 2 bytes, luego los
821                                 // quitamos de la cola de procesado
822                     break;
823
824                     default:                    // Si no comienza por 0xFF es un dato
825                                                 // Si aun no hemos calculado ymax, lo calculamos
826                         if (ymax_valido == false)
827                         {
828                             // Calculamos el valor de ymax
829                             ymax = lienzo.getHeight();
830
831                             // Calculamos el valor de las constantes de escalado vertical
832                             kes[0] = (float) ((-0.1 / 31) * ymax);
833                             kes[1] = (float) ((-0.1 / 62) * ymax);
834                             kes[2] = (float) ((-0.1 / 155) * ymax);
835
836                             yactual1 = ymax/2;  // Para que la 'y' arranque desde el centro
837                             yactual2 = ymax/2;  // de la pantalla
838
839                             yalto1 = (int) (ymax/2);
840                             yalto2 = (int) (ymax/2);
841                             ybajo1 = (int) (ymax/2);
842                             ybajo2 = (int) (ymax/2);
843
844                             // Comprobamos si se ha seleccionado una escala de voltaje distinta a la
845                             // escala por defecto en el Spinner
```

Archivo: FullScreenActivity.java

```
846                             if(escala_voltajes1 == -1)
847                             {
848                                 constante1 = kes[0];
849                             }
850                             else
851                             {
852                                 if((escala_voltajes1 == 0)||(escala_voltajes1 == 3)||
853                                         (escala_voltajes1 == 6)||(escala_voltajes1 == 9))
854                                 {
855                                     constante1 = kes[0];
856                                 }
857                                 else if((escala_voltajes1 == 1)||(escala_voltajes1 == 4)||
858                                         (escala_voltajes1 == 7)||(escala_voltajes1 == 10))
859                                 {
860                                     constante1 = kes[1];
861                                 }
862                                 else
863                                 {
864                                     constante1 = kes[2];
865                                 }
866                             }
867 
868                             if(escala_voltajes2 == -1)
869                             {
870                                 constante2 = kes[0];
871                             }
872                             else
873                             {
874                                 if((escala_voltajes2 == 0)||(escala_voltajes2 == 3)||
875                                         (escala_voltajes2 == 6)||(escala_voltajes2 == 9))
876                                 {
877                                     constante2 = kes[0];
878                                 }
879                                 else if((escala_voltajes2 == 1)||(escala_voltajes2 == 4)||
880                                         (escala_voltajes2 == 7)||(escala_voltajes2 == 10))
881                                 {
882                                     constante2 = kes[1];
883                                 }
884                                 else
885                                 {
886                                     constante2 = kes[2];
887                                 }
888                             }
889 
890                             ymax_valido = true;     // Marcamos ymax como 'calculado'
891                         }
892 
893                         // Falta asignar valores para comparar frecuencias
894 
895                         // Asigno primera posicion al Path
896                         if (i == 0)
897                         {
898                             xactual = 0;
899                             lienzo.mPath.moveTo(xactual, yactual1);
900                             lienzo.mPath2.moveTo(xactual,yactual2);
901 
902                             frecuenciaConseguida1 = false;
903                             frecuenciaConseguida2 = false;
904                             maxEncontrado = false;
905                             minEncontrado = false;
906                             maxEncontrado2 = false;
907                             minEncontrado2 = false;
908 
909                             try
910                             {
```

Archivo: FullScreenActivity.java

```
                                intermedio[0] = escala_voltajes1;
                                intermedio[1] = escala_voltajes2;
                                intermedio[2] = escala_tiempos;
                                intermedio[3] = 0;
                                intermedio[4] = (int) (ymax / 2);
                                intermedio[5] = 0;
                                intermedio[6] = (int) (ymax / 2);
                            }
                            catch (Exception e)
                            {
                                Log.d("Error", "Asignando primeros valores");
                            }
                        }
                        else
                        {
                            if(fin_pantalla == false)
                            {
                                // Asignamos puntos nuevos
                                xactual = xactual + tasa_avance[escala_tiempos];
                                dato1 = composeInt(buffer[i], buffer[i + 1]);
                                dato2 = composeInt(buffer[i + 2], buffer[i + 3]);
                                yactual1 = ((constante1 * (dato1 - 512)) + (ymax / 2));
                                yactual2 = ((constante2 * (dato2 - 512)) + (ymax / 2));

                                if(frecuenciaConseguida1 == false)
                                {
                                    if(maxEncontrado == false)
                                    {
                                        if((yactual1 < composeInt(buffer[i + 4], buffer[i + 5])) &&
                                                (composeInt(buffer[i + 4], buffer[i + 5]) >
                                                        composeInt(buffer[i + 8], buffer[i + 9])))
                                        {
                                            frec11 = xactual + tasa_avance[escala_tiempos];
                                            maxEncontrado = true;
                                        }
                                    }
                                    if(minEncontrado == false)
                                    {
                                        if((yactual1 > composeInt(buffer[i + 4], buffer[i + 5])) &&
                                                (composeInt(buffer[i + 4], buffer[i + 5]) <
                                                        composeInt(buffer[i + 8], buffer[i + 9])))
                                        {
                                            frec21 = xactual + tasa_avance[escala_tiempos];
                                            minEncontrado = true;
                                        }
                                    }
                                    if(minEncontrado == true && maxEncontrado == true)
                                    {
                                        if(frec21 < frec11)
                                        {
                                            valor21 = frec21;
                                            frec21 = frec11;
                                            frec11 = valor21;
                                        }
                                        frecuenciaConseguida1 = true;
                                    }
                                }
                                if(frecuenciaConseguida2 == false)
                                {
                                    if(maxEncontrado2 == false)
                                    {
                                        if((yactual2 < composeInt(buffer[i + 6], buffer[i + 7])) &&
                                                (composeInt(buffer[i + 6], buffer[i + 7]) >
                                                        composeInt(buffer[i + 10], buffer[i + 11])))
                                        {
```

Archivo: FullScreenActivity.java

```
                                    frec12 = xactual + tasa_avance[escala_tiempos];
                                    maxEncontrado2 = true;
                                }
                            }
                            if(minEncontrado2 == false)
                            {
                                if((yactual2 > composeInt(buffer[i + 6], buffer[i + 7])) &&
                                        (composeInt(buffer[i + 6], buffer[i + 7]) <
                                                composeInt(buffer[i + 10], buffer[i + 11])))
                                {
                                    frec22 = xactual + tasa_avance[escala_tiempos];
                                    minEncontrado2 = true;
                                }
                            }
                            if(minEncontrado2 == true && maxEncontrado2 == true)
                            {
                                if(frec22 < frec12)
                                {
                                    valor22 = frec22;
                                    frec22 = frec12;
                                    frec12 = valor22;
                                }
                                frecuenciaConseguida2 = true;
                            }
                        }

                        try
                        {
                            intermedio[i+3] = xactual;
                            intermedio[i+4] = dato1;
                            intermedio[i+5] = xactual;
                            intermedio[i+6] = dato2;
                        }
                        catch (Exception e)
                        {
                            Log.d("Error", "Asignando nuevos valores");
                        }

                        // Asignamos un nuevo segmento al Path
                        lienzo.mPath.lineTo(xactual, yactual1);
                        lienzo.mPath2.lineTo(xactual,yactual2);

                        // Comprobamos valores maximos y minimos de amplitud
                        if(yactual1 > yalto1)
                        {
                            yalto1 = (int) yactual1;
                        }
                        if(yactual1 < ybajo1)
                        {
                            ybajo1 = (int) yactual1;
                        }
                        if(yactual2 > yalto2)
                        {
                            yalto2 = (int) yactual2;
                        }
                        if(yactual2 < ybajo2)
                        {
                            ybajo2 = (int) yactual2;
                        }
                    }
                }

                if((xactual >= xmax) && (fin_pantalla == false))
                {
                    fin_pantalla = true;
```

Archivo: FullScreenActivity.java

```
1041
1042                            // Pintamos las muestras
1043                            lienzo.post(new Runnable()
1044                            {
1045                                public void run()
1046                                {
1047                                    xactual = 0;
1048                                    lienzo.limpiar();
1049                                    DecimalFormat df = new DecimalFormat("#.00");
1050                                    if((byte) (status & 0x03) == (byte) 0x01)
1051                                    {
1052                                        lienzo.drawCanvas.drawPath(lienzo.mPath, lienzo.drawPaint1);
1053                                        lienzo.mPath.reset();
1054                                        lienzo.mPath.moveTo(xactual, yactual1);
1055
1056                                        numberAsString = df.format(((yalto1-ybajo1) * 10 *
1057                                                factorAmp[escala_voltajes1]) / ymax);
1058                                        tv_amp1_val.setText(numberAsString +
1059                                                coletillaAmp[escala_voltajes1]);
1060                                        if((frec21 - frec11) <= 0)
1061                                        {       // No podemos calcular bien la frecuencia
1062                                            tv_frec1_val.setText("Unknown");
1063                                        }
1064                                        else
1065                                        {
1066                                            numberAsString3 = df.format(1 /
1067                                                    (((frec21 - frec11)*2*
1068                                                            factorFrec[escala_tiempos]) / xmax));
1069                                            tv_frec1_val.setText(numberAsString3 +
1070                                                    coletillaFrec[escala_tiempos]);
1071                                        }
1072                                        tv_amp2_val.setText("");
1073                                        tv_frec2_val.setText("");
1074                                    }
1075                                    else if((byte) (status & 0x03) == (byte) 0x02)
1076                                    {
1077                                        lienzo.drawCanvas.drawPath(lienzo.mPath2,lienzo.drawPaint2);
1078                                        lienzo.mPath.reset();
1079                                        lienzo.mPath.moveTo(xactual, yactual2);
1080
1081                                        numberAsString2 = df.format(((yalto2-ybajo2) * 10 *
1082                                                factorAmp[escala_voltajes2]) / ymax);
1083                                        tv_amp2_val.setText(numberAsString2 +
1084                                                coletillaAmp[escala_voltajes2]);
1085                                        if((frec22 - frec12) <= 0)
1086                                        {       // No podemos calcular bien la frecuencia
1087                                            tv_frec2_val.setText("Unknown");
1088                                        }
1089                                        else
1090                                        {
1091                                            numberAsString4 = df.format(1 /
1092                                                    (((frec22 - frec12)*
1093                                                            factorFrec[escala_tiempos]) / xmax));
1094                                            tv_frec2_val.setText(numberAsString4 +
1095                                                    coletillaFrec[escala_tiempos]);
1096                                        }
1097                                        tv_amp1_val.setText("");
1098                                        tv_frec1_val.setText("");
1099                                    }
1100                                    else if((byte) (status & 0x03) == (byte) 0x03)
1101                                    {
1102                                        lienzo.drawCanvas.drawPath(lienzo.mPath, lienzo.drawPaint1);
1103                                        lienzo.drawCanvas.drawPath(lienzo.mPath2,lienzo.drawPaint2);
1104                                        lienzo.mPath.reset();
1105                                        lienzo.mPath.moveTo(xactual, yactual1);
```

Archivo: FullScreenActivity.java

```
1106                                    lienzo.mPath.reset();
1107                                    lienzo.mPath.moveTo(xactual, yactual2);
1108
1109                                    numberAsString = df.format(((yalto1-ybajo1) * 10 *
1110                                            factorAmp[escala_voltajes1]) / ymax);
1111                                    tv_amp1_val.setText(numberAsString +
1112                                            coletillaAmp[escala_voltajes1]);
1113                                    if((frec21 - frec11) <= 0)
1114                                    {       // No podemos calcular bien la frecuencia
1115                                        tv_frec1_val.setText("Unknown");
1116                                    }
1117                                    else
1118                                    {
1119                                        numberAsString3 = df.format(1 /
1120                                                (((frec21 - frec11)*
1121                                                        factorFrec[escala_tiempos]) / xmax));
1122                                        tv_frec1_val.setText(numberAsString3 +
1123                                                coletillaFrec[escala_tiempos]);
1124                                    }
1125
1126                                    numberAsString2 = df.format(((yalto2-ybajo2) * 10 *
1127                                            factorAmp[escala_voltajes2]) / ymax);
1128                                    tv_amp2_val.setText(numberAsString2 +
1129                                            coletillaAmp[escala_voltajes2]);
1130                                    if((frec22 - frec12) <= 0)
1131                                    {       // No podemos calcular bien la frecuencia
1132                                        tv_frec2_val.setText("Unknown");
1133                                    }
1134                                    else
1135                                    {
1136                                        numberAsString4 = df.format(1 /
1137                                                (((frec22 - frec12)*
1138                                                        factorFrec[escala_tiempos]) / xmax));
1139                                        tv_frec2_val.setText(numberAsString4 +
1140                                                coletillaFrec[escala_tiempos]);
1141                                    }
1142                                }
1143
1144                                pintado = true;
1145
1146                                yalto1 = (int) (ymax/2);
1147                                yalto2 = (int) (ymax/2);
1148                                ybajo1 = (int) (ymax/2);
1149                                ybajo2 = (int) (ymax/2);
1150                            }
1151                        });
1152                    }
1153
1154                    i += 4; // Todos los datos que nos envian son de 4 bytes por muestra (dos
1155                            // por cada canal), luego los quitamos de la cola de procesado
1156                break;
1157            }
1158        }
1159
1160        if(fin_pantalla == true)
1161        {
1162            while (pintado == false)
1163            {
1164            }
1165
1166            try
1167            {
1168                copiaDatos = intermedio;
1169            }
1170            catch (Exception e)
```

Archivo: FullScreenActivity.java

```
1171                {
1172                    Log.d("Error", "Copiando buffers");
1173                }
1174
1175                comando[0] = (byte) 0xF3;   // Avisamos al dsPIC que ya hemos pintado una pantalla
1176                comando[1] = status;        // De paso le enviamos el status actual
1177                enviaDato(comando,0,2);
1178
1179                fin_pantalla = false;
1180                pintado = false;
1181            }
1182        }
1183    }
1184
1185    // ********************************************************************************************
1186
1187    // Método para el botón "Funciones"
1188    public void Funciones(View view)
1189    {
1190        //Desplagar un cuadro de dialogo con opciones de funciones:
1191        //1 - Guardar captura de pantalla
1192        //2 - Guardar archivo de datos
1193        //3 - Etc.
1194
1195        final Dialog menuFunciones = new Dialog(this);
1196        menuFunciones.requestWindowFeature(Window.FEATURE_NO_TITLE);
1197        menuFunciones.setContentView(R.layout.funciones_menu);
1198
1199        TextView tv_func_aceptar = (TextView)menuFunciones.findViewById(R.id.tv_func_aceptar);
1200        final CheckBox check_screenshot = (CheckBox)menuFunciones.findViewById(R.id.screenshot);
1201        final CheckBox check_data = (CheckBox)menuFunciones.findViewById(R.id.data_storage);
1202        final CheckBox check_otras = (CheckBox)menuFunciones.findViewById(R.id.otras);
1203
1204        tv_func_aceptar.setOnClickListener(new View.OnClickListener()
1205        {
1206            @Override
1207            public void onClick(View v)
1208            {
1209                if(check_screenshot.isChecked() == true)
1210                {
1211                    // En el caso de querer hacer una captura a toda la pantalla, le pasamos
1212                    // 'getWindow().getDecorView().getRootView()' en vez de 'lienzo' (sin comillas)
1213                    Bitmap capturaCanvas = takeScreenShot(lienzo);
1214                    guardar(capturaCanvas);
1215                }
1216                if(check_data.isChecked() == true)
1217                {
1218                    salvarDatos(copiaDatos);
1219                }
1220                if(check_otras.isChecked() == true)
1221                {
1222                    otrasFunciones();
1223                }
1224                menuFunciones.dismiss();
1225            }
1226        });
1227
1228        menuFunciones.show();
1229    }
1230
1231    // Funcion para tomar una captura de pantalla de la representacion de las señales
1232    public Bitmap takeScreenShot(View view) {
1233        // Configuramos para que la view almacene la cache en una imagen
1234        view.setDrawingCacheEnabled(true);
1235        view.setDrawingCacheQuality(View.DRAWING_CACHE_QUALITY_HIGH);
```

Archivo: FullScreenActivity.java

```
1236        view.buildDrawingCache();
1237        if (view.getDrawingCache() == null)
1238        {
1239            return null; // Verificamos antes de que no sea null
1240        }
1241        // utilizamos esa cache, para crear el bitmap que tendra la imagen de la view actual
1242        Bitmap captura = Bitmap.createBitmap(view.getDrawingCache());
1243        view.setDrawingCacheEnabled(false);
1244        view.destroyDrawingCache();
1245        return captura;
1246    }
1247
1248    // Funcion para guardar la captura de pantalla
1249    private void guardar(Bitmap imagenCanvas)
1250    {
1251        String root = Environment.getExternalStoragePublicDirectory(
1252                Environment.DIRECTORY_PICTURES).toString();
1253        File directorio = new File(root + "/OscAndroid");
1254
1255        if (!directorio.exists())
1256        {
1257            directorio.mkdirs();
1258        }
1259        Date date = new Date();
1260        DateFormat hourdateFormat = new SimpleDateFormat("yyyyMMdd_HHmmss");
1261        String historial = hourdateFormat.format(date);
1262
1263        String nombre = "OscAndroid_" + historial + ".png";
1264        File file = new File(directorio, nombre);
1265        if (file.exists())
1266        {
1267            file.delete();
1268        }
1269        try {
1270            FileOutputStream out = new FileOutputStream(file);
1271            imagenCanvas.compress(Bitmap.CompressFormat.PNG, 100, out);
1272            out.flush();
1273            out.close();
1274            Toast.makeText(getApplicationContext(),
1275                    "Imagen guardada en la Galeria",
1276                    Toast.LENGTH_SHORT).show();
1277
1278        } catch (Exception e) {
1279            e.printStackTrace();
1280        }
1281
1282        MediaScannerConnection.scanFile(this, new String[]{file.toString()}, null,
1283                new MediaScannerConnection.OnScanCompletedListener() {
1284                    public void onScanCompleted(String path, Uri uri) {
1285                    }
1286                });
1287    }
1288
1289    // Funcion para guardar la informacion de captura en un archivo de texto
1290    private void salvarDatos(int[] buffer)
1291    {
1292        Date date = new Date();
1293        DateFormat hourdateFormat = new SimpleDateFormat("yyyyMMdd_HHmmss");
1294        String historial = hourdateFormat.format(date);
1295
1296        String nombre = "OscAndroid_" + historial + ".txt";
1297
1298        String estado = Environment.getExternalStorageState();
1299
1300        if(estado.equals(Environment.MEDIA_MOUNTED))
```

Archivo: FullScreenActivity.java

```
1301        {
1302            sdDisponible = true;
1303            sdAccesoEscritura = true;
1304        }
1305        else if(estado.equals(Environment.MEDIA_MOUNTED_READ_ONLY))
1306        {
1307            sdDisponible = true;
1308            sdAccesoEscritura = false;
1309        }
1310        else
1311        {
1312            sdDisponible = false;
1313            sdAccesoEscritura = false;
1314        }
1315
1316        if(sdDisponible && sdAccesoEscritura)
1317        {
1318            try
1319            {
1320                File ruta_sd = Environment.getExternalStorageDirectory();
1321                File f = new File(ruta_sd.getAbsolutePath(), nombre);
1322
1323                OutputStreamWriter fout =
1324                        new OutputStreamWriter(
1325                                new FileOutputStream(f));
1326
1327                fout.write(Arrays.toString(buffer));
1328                fout.close();
1329
1330                Toast.makeText(this, "Archivo guardado", Toast.LENGTH_SHORT).show();
1331            }
1332            catch (IOException e)
1333            {
1334                Toast.makeText(this, "Error al guardar archivo",
1335                        Toast.LENGTH_SHORT).show();
1336            }
1337        }
1338    }
1339
1340    // Otras funciones
1341    private void otrasFunciones()
1342    {
1343        Toast.makeText(this,"Coming Soon...", Toast.LENGTH_SHORT).show();
1344    }
1345 }
1346
1347 //***********************************************************************************************
1348 // Fin de archivo
1349 //***********************************************************************************************
```

Archivo: activity_fullscreen.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#2F2F31"
    tools:context=".FullscreenActivity">

    <TextView
        android:id="@+id/tv_frec1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginLeft="16dp"
        android:layout_marginBottom="16dp"
        android:text="@string/tv_frec1"
        android:textColor="#FFFFFFFF"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toStartOf="parent" />

    <TextView
        android:id="@+id/tv_amp1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginLeft="16dp"
        android:layout_marginBottom="12dp"
        android:text="@string/tv_amp1"
        android:textColor="#FFFFFFFF"
        app:layout_constraintBottom_toTopOf="@+id/tv_frec1"
        app:layout_constraintStart_toStartOf="parent" />

    <TextView
        android:id="@+id/tv_amp1_val"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="12dp"
        android:layout_marginLeft="12dp"
        android:layout_marginBottom="12dp"
        android:text="TextView"
        android:textColor="#00BCD4"
        app:layout_constraintBottom_toTopOf="@+id/tv_frec1_val"
        app:layout_constraintStart_toEndOf="@+id/tv_amp1" />

    <TextView
        android:id="@+id/tv_frec1_val"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="12dp"
        android:layout_marginLeft="12dp"
        android:layout_marginBottom="16dp"
        android:text="TextView"
        android:textColor="#00BCD4"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintStart_toEndOf="@+id/tv_frec1" />

    <ToggleButton
        android:id="@+id/boton_ch1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="92dp"
        android:layout_marginLeft="92dp"
        android:layout_marginBottom="16dp"
```

Archivo: activity_fullscreen.xml

```
66          android:checked="false"
67          android:textOff="@string/toggle1_off"
68          android:textOn="@string/toggle1_on"
69          app:layout_constraintBottom_toBottomOf="parent"
70          app:layout_constraintStart_toEndOf="@+id/tv_amp1" />
71
72      <TextView
73          android:id="@+id/tv_amp2"
74          android:layout_width="wrap_content"
75          android:layout_height="wrap_content"
76          android:layout_marginEnd="12dp"
77          android:layout_marginRight="12dp"
78          android:layout_marginBottom="12dp"
79          android:text="@string/tv_amp2"
80          android:textColor="#FFFFFFFF"
81          app:layout_constraintBottom_toTopOf="@+id/tv_frec2"
82          app:layout_constraintEnd_toStartOf="@+id/tv_frec2_val" />
83
84      <TextView
85          android:id="@+id/tv_frec2"
86          android:layout_width="wrap_content"
87          android:layout_height="wrap_content"
88          android:layout_marginEnd="12dp"
89          android:layout_marginRight="12dp"
90          android:layout_marginBottom="16dp"
91          android:text="@string/tv_frec2"
92          android:textColor="#FFFFFFFF"
93          app:layout_constraintBottom_toBottomOf="parent"
94          app:layout_constraintEnd_toStartOf="@+id/tv_frec2_val" />
95
96      <TextView
97          android:id="@+id/tv_amp2_val"
98          android:layout_width="wrap_content"
99          android:layout_height="wrap_content"
100         android:layout_marginBottom="12dp"
101         android:text="TextView"
102         android:textColor="#FFC107"
103         app:layout_constraintBottom_toTopOf="@+id/tv_frec2_val"
104         app:layout_constraintStart_toStartOf="@+id/tv_frec2_val" />
105
106     <TextView
107         android:id="@+id/tv_frec2_val"
108         android:layout_width="wrap_content"
109         android:layout_height="wrap_content"
110         android:layout_marginEnd="24dp"
111         android:layout_marginRight="24dp"
112         android:layout_marginBottom="16dp"
113         android:text="TextView"
114         android:textColor="#FFC107"
115         app:layout_constraintBottom_toBottomOf="parent"
116         app:layout_constraintEnd_toStartOf="@+id/boton_ch2" />
117
118     <ToggleButton
119         android:id="@+id/boton_ch2"
120         android:layout_width="wrap_content"
121         android:layout_height="wrap_content"
122         android:layout_marginEnd="16dp"
123         android:layout_marginRight="16dp"
124         android:layout_marginBottom="16dp"
125         android:textOff="@string/toggle2_off"
126         android:textOn="@string/toggle2_on"
127         app:layout_constraintBottom_toBottomOf="parent"
128         app:layout_constraintEnd_toStartOf="@+id/Lado_derecho" />
129
130     <view
```

Archivo: activity_fullscreen.xml

```
        android:id="@+id/lienzo"
        class="com.example.oscandroid.Lienzo"
        id="@+id/view"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_marginStart="16dp"
        android:layout_marginLeft="16dp"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="16dp"
        android:layout_marginRight="16dp"
        android:layout_marginBottom="16dp"
        android:background="@drawable/fondolienzo"
        app:layout_constraintBottom_toTopOf="@+id/tv_amp1"
        app:layout_constraintEnd_toStartOf="@+id/Lado_derecho"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <LinearLayout
        android:id="@+id/Lado_derecho"
        android:layout_width="120dp"
        android:layout_height="0dp"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="16dp"
        android:layout_marginRight="16dp"
        android:layout_marginBottom="8dp"
        android:orientation="vertical"
        app:layout_constraintBottom_toTopOf="@+id/boton_func"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <TextView
            android:id="@+id/tv_vert1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/tv_vert1"
            android:textColor="#FFFFFFFF" />

        <Spinner
            android:id="@+id/spinner_vert1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:popupBackground="#515153" />

        <TextView
            android:id="@+id/tv_vert2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/tv_vert2"
            android:textColor="#FFFFFFFF" />

        <Spinner
            android:id="@+id/spinner_vert2"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:popupBackground="#515153" />

        <TextView
            android:id="@+id/tv_tiempos"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
```

Archivo: activity_fullscreen.xml

```
196             android:text="@string/tv_tiempos"
197             android:textColor="#FFFFFFFF" />
198 
199         <Spinner
200             android:id="@+id/spinner_tiempos"
201             android:layout_width="match_parent"
202             android:layout_height="wrap_content"
203             android:layout_weight="1"
204             android:popupBackground="#515153" />
205 
206         <TextView
207             android:id="@+id/tv_trigger"
208             android:layout_width="match_parent"
209             android:layout_height="wrap_content"
210             android:layout_weight="1"
211             android:text="@string/tv_trigger"
212             android:textColor="#FFFFFFFF" />
213 
214         <Spinner
215             android:id="@+id/spinner_trigger"
216             android:layout_width="match_parent"
217             android:layout_height="wrap_content"
218             android:layout_weight="1"
219             android:popupBackground="#515153" />
220 
221         <LinearLayout
222             android:id="@+id/Espacio"
223             android:layout_width="match_parent"
224             android:layout_height="wrap_content"
225             android:layout_weight="5"
226             android:orientation="horizontal">
227 
228         </LinearLayout>
229 
230     </LinearLayout>
231 
232     <Button
233         android:id="@+id/boton_func"
234         android:layout_width="wrap_content"
235         android:layout_height="wrap_content"
236         android:layout_marginEnd="16dp"
237         android:layout_marginRight="16dp"
238         android:layout_marginBottom="16dp"
239         android:layout_weight="2"
240         android:onClick="Funciones"
241         android:text="@string/boton_func"
242         android:textAllCaps="false"
243         android:textSize="16sp"
244         app:layout_constraintBottom_toBottomOf="parent"
245         app:layout_constraintEnd_toEndOf="parent" />
246 
247 </android.support.constraint.ConstraintLayout>
```

Archivo: Lienzo.java

```
/*
 * Proyecto:    OscAndroid_Software
 * Archivo:     Lienzo.java
 * Autor:       David Ortiz de Latierro Delgado
 * Rev:         1.0
 */

package com.example.oscandroid;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.PorterDuff;
import android.util.AttributeSet;
import android.util.Log;
import android.view.View;

// Objeto para crear el lienzo sobre el que dibujar las señales
public class Lienzo extends View
{
    // Pinceles para dibujar las señales de los canales 1 y 2
    public Paint drawPaint1, drawPaint2, canvasPaint;

    // Colores para los canales 1 y 2
    private int color_ch1 = 0xFF00BCD4;
    private int color_ch2 = 0xFFFFC107;

    // Canvas
    public Canvas drawCanvas;

    // Canvas para guardar
    private Bitmap canvasBitmap;

    // Variables para representar las lineas
    public float yantes1 = 0;
    public float yantes2 = 0;
    public float yahora1 = 0;
    public float yahora2 = 0;
    public float xantes = 0;
    public float xahora = 0;

    // Variable para guardar el/los canal(es) activo(s)
    public int canal = 0;

    // Variable para ir guardando los puntos y luego unirlos mediante una funcion de trazado
    public Path mPath;
    public Path mPath2;

    // ********************************************************************************************

    public Lienzo(Context context, AttributeSet attrs)
    {
        super(context, attrs);
        setupDrawing();
    }

    // ********************************************************************************************
    // Función para configurar los pinceles
    private void setupDrawing()
    {
        // Creamos los Paths

        mPath = new Path();
```

Archivo: Lienzo.java

```
66          mPath2 = new Path();
67
68          // Configuramos caracteristicas de los pinceles
69
70          canvasPaint = new Paint(Paint.DITHER_FLAG);
71
72          drawPaint1 = new Paint();
73          drawPaint1.setColor(color_ch1);
74          drawPaint1.setAntiAlias(true);
75          drawPaint1.setStrokeWidth(5);
76          drawPaint1.setStyle(Paint.Style.STROKE);
77          drawPaint1.setStrokeJoin(Paint.Join.ROUND);
78          drawPaint1.setStrokeCap(Paint.Cap.ROUND);
79
80          drawPaint2 = new Paint();
81          drawPaint2.setColor(color_ch2);
82          drawPaint2.setAntiAlias(true);
83          drawPaint2.setStrokeWidth(5);
84          drawPaint2.setStyle(Paint.Style.STROKE);
85          drawPaint2.setStrokeJoin(Paint.Join.ROUND);
86          drawPaint2.setStrokeCap(Paint.Cap.ROUND);
87      }
88
89      // ********************************************************************************************
90      // Asignamos un tamaño a la vista sobre la que pintamos
91      @Override
92      protected void onSizeChanged(int w, int h, int oldw, int oldh)
93      {
94          super.onSizeChanged(w, h, oldw, oldh);
95          canvasBitmap = Bitmap.createBitmap(w, h, Bitmap.Config.ARGB_8888);
96          drawCanvas = new Canvas(canvasBitmap);
97      }
98
99      // ********************************************************************************************
100     // Este método es el que pinta en el canvas. Hay que llamar a repintar constantemente desde un
101     // invalidate, el cual estará en una funcion que recoja los valores de x e y para las lineas
102     // A esa funcion habra que llamarla desde el FullScreen Activity
103     // Otra opción sería crear un drawPath de la clase Path, de manera que le fuesemos dando los
104     // puntos que hay que ir pintando y luego, llamar a pintar una vez esté el trazo completo.
105     // Esta ultima opcion sería más rapida, ya que en vez de llamar a repintar tantas veces como
106     // puntos en pantalla tengamos, solo llamamos a repintar una vez, para que dibuje el trazo.
107     // Pintamos sobre el canvas
108     @Override
109     protected void onDraw(Canvas canvas)
110     {
111         canvas.drawBitmap(canvasBitmap,0,0,canvasPaint);
112
113         if(canal == 1)
114         {
115             canvas.drawPath(mPath,drawPaint1);
116         }
117         else if(canal == 2)
118         {
119             canvas.drawPath(mPath2,drawPaint2);
120         }
121         else if(canal == 3)
122         {
123             canvas.drawPath(mPath,drawPaint1);
124             canvas.drawPath(mPath2,drawPaint2);
125         }
126     }
127
128     // ********************************************************************************************
129     // Funcion que sirve para borrar la pantalla
130     // En el caso de que esta funcion no nos sirva, la opcion mas facil para borrar la pantalla
```

Archivo: Lienzo.java

```
131     // (si estamos usando un Path) sería llamar a pintar ese Path, pero con color alpha = 0, es
132     // decir, transparente.
133     public void limpiar()
134     {
135         drawCanvas.drawColor(0, PorterDuff.Mode.CLEAR);
136         //invalidate();
137     }
138 
139 }
140 
141 //*********************************************************************************************
142 // Fin de archivo
143 //*********************************************************************************************
```

## G. Contenido del CD

En este anexo se indica el contenido del CD en el que se presenta este Trabajo de Fin de Grado. Incluye todos los archivos necesarios para la programación de los componentes y la fabricación/edición de nuevos prototipos del sistema. Todos los contenidos son accesibles a través del CD, incluyendo el presente documento en formato digital.

Todos los archivos están organizados en carpetas según la parte del sistema a la que pertenecen. El esquema de archivos es el siguiente:

- Directorio raíz
    - Documento en versión digital
    - Carpeta *"Hardware"*
        - Archivo de diseño del circuito para *Proteus* (ext. *DSN*)
        - Archivo de diseño del *layout* del circuito para *Proteus* (ext. LYT)
        - Ficheros *Gerber* para fabricación
        - Carpeta *"Componentes"*
            - Hojas de especificación de todos los elementos que componen el circuito
    - Carpeta *"Firmware"*
        - Proyecto para *MPLAB* que contiene los archivos de código del programa para el Hardware
    - Carpeta *"Software"*
        - Proyecto para *Android Studio* con los archivos de código de la aplicación para *Android*