



UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
DE TELECOMUNICACIÓN

**MÁSTER UNIVERSITARIO EN INGENIERÍA DE
TELECOMUNICACIÓN**

**DESARROLLO Y PUESTA EN FUNCIONAMIENTO DE UN
SISTEMA DE ADQUISICIÓN EEG CON CAPACIDADES DE
PROCESADO**

Autor: Javier Benavides Caro

Director: Juan Manuel López Navarro

Madrid, 2018

**DESARROLLO Y PUESTA EN FUNCIONAMIENTO DE UN
SISTEMA DE ADQUISICIÓN EEG CON CAPACIDADES DE
PROCESADO**

Autor: Javier Benavides Caro
Director: Juan Manuel López Navarro

DEPARTAMENTO DE INGENIERÍA TELEMÁTICA Y ELECTRÓNICA

Agradecimientos

La realización de este proyecto no habría sido posible sin la ayuda de ciertas personas que me han apoyado durante toda su realización.

En primer lugar al tutor, que ha proporcionado su ayuda en los momentos más complicados y siempre ha mantenido un grado de paciencia y exigencia alto, dando lugar a una calidad en el trabajo sin duda superior a la que habría conseguido alcanzar por mis propios medios.

A Jesús Alonso por su ayuda durante el comienzo del proyecto, asentando unas bases y explicando todos aquellos conceptos desconocidos hasta ese momento.

Por supuesto a mis padres, pues su apoyo y esfuerzo me ha permitido estudiar aquello que más me gusta desde el principio hasta este momento y estoy seguro de que este camino no ha estado exento de sacrificios por su parte.

Merece especial mención Nerea, que ha estado presente en todo momento ayudando y compartiendo sus opiniones siempre que ha sido necesario, sacrificando tiempo libre y horas de sueño sin dudarlo.

Por último a todos mis amigos, tanto de Granada como de Alcobendas.

Resumen

Resumen —

Este proyecto tiene como objetivo el diseño y prototipado de un sistema de adquisición y procesado de electroencefalogramas. El sistema desarrollado cuenta con un microcontrolador de la familia ARM Cortex M4 como núcleo de procesamiento y gestión de dispositivos. Este microcontrolador se encargará de la configuración y comunicación con el ADC (ADS1299), chip que realiza la adquisición de los datos que conforman el electroencefalograma. Además, el ARM procesará y filtrará la información obtenida. Finalmente se transmitirá esta información a un ordenador utilizando una de las dos interfaces inalámbricas disponibles (WiFi). Se ha realizado un software para el ordenador que permite cambiar la configuración del sistema de adquisición y representar los datos procesados en gráficas.

Palabras clave — ADC, adquisición, filtrado, microcontrolador, PCB, interfaz.

Abstract

Abstract —

The objective of this project is the design and prototyping of an electroencephalogram acquisition and processing system. The developed system has a microcontroller of the ARM Cortex M4 family as the core of processing and device management. This microcontroller will be responsible for the configuration and communication with the ADC (ADS1299), chip that performs the acquisition of the data that conforms the electroencephalogram. In addition, the ARM will process and filter the information obtained. Eventually this information will be transmitted to a computer using one of the two available wireless interfaces (WiFi). A user interface created using LabView will be able to change the configuration and represent all the adquired data.

Key words — ADC, adquisition, filtering, microcontroller, PCB, interface.

Índice general

1. Introducción	1
1.1. Alcance y estructura del proyecto	3
1.2. Base teórica	4
1.3. Tipos de electrodos	7
1.3.1. Electrodos húmedos	7
1.3.2. Electrodos secos	9
2. Estado del Arte	11
2.1. Dispositivos similares	12
2.1.1. Proyectos personales	12
2.1.2. Proyectos docentes	14
2.1.3. Sistemas comerciales	15
2.2. Comparativa	17
3. Diseño	19
3.1. Diseño base	19
3.1.1. Adquisición de datos	20
3.1.2. Transmisión de datos	21
3.2. Diseño final	24
3.2.1. Selección del microcontrolador	25
3.2.2. Alimentación	27
3.3. Circuito electrónico y esquemáticos	29
3.3.1. Circuito de alimentación	30
3.3.2. Microcontrolador	31
3.3.3. Interfaz inalámbrica	33
4. Implementación de la PCB	35
4.1. Limitaciones del fabricante	36
4.2. Componentes y librerías	39
4.2.1. Asignación de <i>footprint</i>	39
4.3. PCBnew	42
4.4. Diseño final de la PCB	44
4.4.1. Alimentación	45
4.4.2. Microcontrolador	45

ÍNDICE GENERAL

4.4.3. Interfaz inalámbrica	46
4.5. Resultados	48
4.5.1. Previsualización 3D vs resultado final	49
5. Implementación del software	51
5.1. Microcontrolador STM32F4	51
5.1.1. Configuración inicial	53
5.1.2. Implementación del firmware	59
5.2. Interfaz inalámbrica	76
5.2.1. Arduino	76
5.2.2. Configuración	77
5.2.3. Firmware	78
5.3. LabView	85
5.3.1. Interfaz gráfica	85
5.3.2. Comunicación WiFi	87
5.3.3. Conversión de datos	88
5.3.4. Alternativa UDP	88
6. Resultados	89
6.1. Funcionamiento del ADS	89
6.2. Medidas sobre señales reales	91
6.3. Filtrado de la señal	92
6.4. Medida de un EEG real	96
6.5. Transmisión de datos por WiFi	97
7. Conclusiones	99
7.1. Futuras mejoras	100
Bibliografía	105
Apéndices	109
A. Bill of Materials (BOM)	111
B. Impacto ético, económico, social y ambiental	113
C. Esquemático del proyecto	115
D. Firmware del ESP12-E	119
E. Firmware del STM32F4	129

Índice de figuras

1.1. Ejemplo de anatomía humana	2
1.2. Ejemplo de ECG	3
1.3. Sinapsis neuronal [2]	4
1.4. Ondas delta [1]	4
1.5. Ondas theta [1]	5
1.6. Ondas alfa [1]	5
1.7. Ondas beta [1]	6
1.8. Esquema de colocación de los electrodos [1]	6
1.9. Electrodos de en un sistema de adquisición [1]	7
1.10. Esquema de un electrodo húmedo [1]	8
1.11. Medida de un EEG usando electrodos húmedos [1]	8
1.12. Electrodo seco real(izquierda) y representación 3D (derecha) [1]	9
2.1. Esquema electrónico casero para la captura de un EEG [3]	12
2.2. Elementos que conforman OpenEEG [4]	13
2.3. Beanie en funcionamiento [5]	14
2.4. Imagen promocionando el producto	15
2.5. CAPTION	16
2.6. Módulo Ganglion montado sobre un casco [6]	16
3.1. Placa final del proyecto base	19
3.2. Esquema del proyecto base	20
3.3. Convertidor Analógico-Digital ADS1299	20
3.4. ESP8266	22
3.5. Resumen de todas las Entradas/Salidas del ESP12-E [9]	23
3.6. Simblee™ RFD77101 [10]	23
3.7. Esquema del proyecto final	24
3.8. Principales fabricantes contemplados	25
3.9. Placa de desarrollo STM32F4 Discovery	26
3.10. Esquema de alimentación a 3.3V	27
3.11. Esquema de alimentación a 5V	28
3.12. Esquema general del sistema	29
3.13. Esquemático final del circuito de alimentación	30
3.14. Esquemático del microcontrolador	31
3.15. Detalle del circuito asociado al oscilador externo	32

ÍNDICE DE FIGURAS

3.16. Condensadores Bulk (C11) y de desacoplo	32
3.17. Esquemático del ESP-12E	33
3.18. Esquemático del dispositivo Bluetooth Simblee	34
4.1. Gestor de proyectos de KiCad	36
4.2. Tipos de vías	38
4.3. Reglas de diseño en KiCad cumpliendo las restricciones de ITEAD	38
4.4. Resistencia en formato 0603 [17]	39
4.5. <i>Footprint</i> modificado para adaptarlo al ESP12-E	40
4.6. Herramienta cvpcb asociando a un componente su <i>footprint</i>	41
4.7. Panel principal de la herramienta PCBnew	42
4.8. Ejemplo de planos, <i>keep out area</i> y y pistas en la PCB	43
4.9. Distintas partes que componen la PCB. Alimentación (rojo), microcontrolador (blanco), interfaz inalámbrica (azul).	44
4.10. Detalle de la alimentación de la placa	45
4.11. Detalle del microcontrolador y los componentes asociados	46
4.12. Detalle de los módulos WiFi y Bluetooth	47
4.13. <i>Layout</i> en KiCad (a) e impresión final (b) de la capa superior	48
4.14. <i>Layout</i> en KiCad (a) e impresión final (b) de la capa inferior	49
4.15. Modelo 3D del sistema (arriba) y placa final (abajo)	50
5.1. IDEs alternativos disponibles [18]	52
5.2. Ejemplo de IDE alternativo basado en Oracle (System Workbench for STM32)	53
5.3. Inicializador de proyectos STM32CubeMX	53
5.4. Espacio de trabajo del STM32CubeMX	54
5.5. Configuración del reloj del microcontrolador	55
5.6. Configuración de las interfaces de comunicación	56
5.7. Detalle de la configuración de las interfaces de comunicación	56
5.8. Calculadora del consumo del microcontrolador	57
5.9. Resumen del funcionamiento y características del HAL [19]	59
5.10. Estructura del programa	60
5.11. Esquema de comunicaciones SPI [2]	61
5.12. Escritura de dos registros del ADS [7]	63
5.13. Lectura de dos registros del ADS [7]	64
5.14. Modo de uso del comando RDATA	68
5.15. Transmisión de datos en modo continuo	69
5.16. Diagrama de bloques del filtrado de una señal $x[n]$	71
5.17. Respuesta en frecuencia del filtro definido con fir1	72
5.18. Respuesta en frecuencia del filtro FIR definido con FIRFILT [21]	73
5.19. Herramienta de programación a través de UART	75
5.20. Arduino IDE utilizado para la programación del ESP	76
5.21. Importar librerías en Arduino	77
5.22. Instalar librerías para nuevas placas	77

ÍNDICE DE FIGURAS

5.23. Maquina de estados del ESP	78
5.24. Proceso de división y ensamblado de float para su transmisión	80
5.25. Logotipo de LabView	85
5.26. Interfaz de LabView: Representación de la información	85
5.27. Interfaz de LabView: Configuración del ADS	86
5.28. LabView: Transmisión y recepción de datos a través de TCP	87
5.29. Bloques para ignorar errores conocidos	87
5.30. Extracción y conversión de datos	88
5.31. Recepción de datos a través de UDP	88
6.1. Comparativa entre la señal test y su amplitud teórica	90
6.2. Señal de test medida usando LabView	90
6.3. Señal real triangular generada con Arduino	91
6.4. Simulación del efecto del filtro Notch a 50Hz	92
6.5. Implementación real del efecto del filtro Notch a 50Hz	93
6.6. Señal de 5Hz filtrada	93
6.7. Señal de 10Hz filtrada	94
6.8. Señal de 50Hz filtrada	94
6.9. Señal de 65Hz filtrada	95
6.10. EEG abriendo los ojos	96
6.11. EEG con los ojos cerrados	96
6.12. Sistema de medida utilizado para la adquisición del EEG	97
6.13. Comparativa entre los distintos retardos y el porcentaje de paquetes perdidos	98
B.1. Directiva RoHS [35]	113

Índice de tablas

2.1. Comparativa entre los dispositivos	17
3.1. Tabla de registros de la familia ADS [7]	21
3.2. Modos de arranque del ESP12-E [8]	22
3.3. Comparativa entre distintos MCUs	26
3.4. Rangos de alimentación de todos los elementos	27
4.1. Restricciones de ITEAD para la fabricación de una PCB	37
5.1. Equivalencia de voltaje ideal de entrada y código de salida	65
5.2. Comandos del ADS1299	67
5.3. Representación de datos usando client.print()	84
A.1. BOM - Bill of Materials	111

Índice de códigos

5.1.	Ejemplo de generación de código automáticamente	58
5.2.	Transmisión de datos a través de SPI con el STM	62
5.3.	División de variables en otras de menor tamaño	62
5.4.	Ejemplo de definiciones de los registros y funciones del ADS	63
5.5.	Escritura en registros del ADS	64
5.6.	Lectura de registros del ADS	64
5.7.	Decodificación de datos binarios a Voltajes	66
5.8.	Lectura de datos en modo <i>one-shot</i>	68
5.9.	Lectura de datos en modo continuo	69
5.10.	Filtro FIR en STM	71
5.11.	Transmisión de datos a través de SPI con el STM	74
5.12.	Configuración de pines para el uso de SPI	79
5.13.	Variables y tipos de dato específicos	80
5.14.	Variables y tipos de dato específicos	81
5.15.	Conexión del ESP a una red WiFi ya existente	81
5.16.	Creacion de un punto de acceso en el ESP	82
5.17.	Gestión de clientes y peticiones en Arduino	82
5.18.	Gestión de clientes y peticiones en Arduino	82
5.19.	Gestión de clientes y peticiones en Arduino	83
5.20.	Reinicios por <i>watchdog</i>	83
D.1.	ESP:master_fast.ino	119
E.1.	STM32F4:main.c	129
E.2.	STM32F4:main.h	143
E.3.	STM32F4:ADS1299.c	145
E.4.	STM32F4:ADS1299.h	155

1

Introducción

Desde el principio de los tiempos la humanidad se ha esforzado por comprender el entorno que le rodea, aprender de él y usarlo en su propio beneficio para conseguir así hacer su vida más fácil. Por el momento hemos conseguido hacer volar aviones gracias a la observación de los pájaros o crear sistemas de sonar que se asemejan al sistema que utilizan los murciélagos para orientarse. Pero irónicamente, a pesar del esfuerzo invertido, nuestro propio cuerpo sigue albergando secretos que desentrañar que podrían facilitarle la vida a un gran número de personas.

El estudio del **cuerpo humano** ha sido uno de los temas más polémicos y que más ha evolucionado desde que hay registro. Aunque al comienzo estuvo muy marcado por la superstición y la religión, achacando la mayoría de las dolencias y efectos sanadores a la magia, con el paso del tiempo aparecieron personas como Hipócrates y Aristóteles que fueron capaces de aportar un nuevo enfoque basado en la **observación y estudio** de lo que les rodeaba, asentando unas bases que, posteriormente, serían aprovechadas y mejoradas hasta convertirse en lo que conocemos hoy en día como **método científico**.

El proceso de descubrimiento del funcionamiento del cuerpo humano ha seguido diferentes fases a lo largo de la historia. Comparar el cuerpo humano con animales fue uno de los primeros pasos para descubrir cómo estábamos formados por dentro. Posteriormente, aprovechando los cuerpos de personas ya fallecidas se estudió la anatomía humana, permitiendo así crear mapas y dibujos de la estructura del cuerpo humano y sus órganos bastante detallados.

CAPÍTULO 1. INTRODUCCIÓN

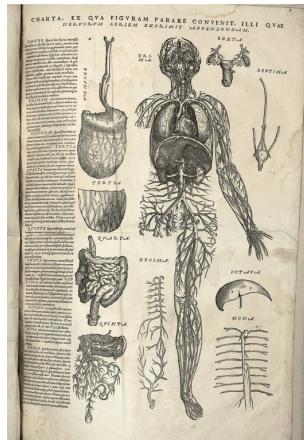


Figura 1.1: Ejemplo de anatomía humana.

Pese a todo, estudiar cuerpos inertes tiene sus limitaciones de modo que durante un tiempo se realizaron vivisecciones para poder comprender mejor como funcionaban todos aquellos órganos, músculos y nervios que ya habían visto con anterioridad. Con el paso del tiempo este sistema fue descartado ya que es una práctica que ponía en peligro la vida del sujeto, haciéndolo pasar por una experiencia terrible en el mejor de los casos.

En la actualidad, gracias al conocimiento acumulado de muchos años y a los **avances en otros campos de la ciencia**, se han desarrollado dispositivos y técnicas que permiten el **estudio en vivo** del comportamiento del cuerpo humano de **forma no invasiva**. Es posible utilizar **ecografías** para ver el estado del corazón, radiografías para diagnosticar un hueso roto e incluso técnicas más avanzadas como la **medicina nuclear** que permiten saber que partes del cerebro se activan frente a determinados estímulos sin necesidad de interactuar físicamente con él.

Si bien todas las técnicas anteriores han supuesto auténticos hitos en la medicina moderna y han permitido diagnosticar un gran número de enfermedades así como mejorar la calidad de vida de muchas personas, la mayoría presenta inconvenientes que hacen improbable su uso a nivel personal o docente debido al tamaño de los equipos necesarios para su realización o el coste muy elevado del procedimiento (sin contar con el conocimiento necesario para la realización correcta de la prueba).

Teniendo en mente esta problemática se han desarrollado dispositivos capaces de medir pequeñas las variaciones de voltaje que se producen en el interior de nuestro cuerpo haciendo uso de unos dispositivos denominados **electrodos**.

De esta forma es posible, con un **coste muy reducido** y un equipamiento relativamente asequible, conseguir inferir que procesos químicos y físicos se están produciendo en el interior de nuestro cuerpo.

Haciendo uso de este sistema se pueden obtener registros sobre diferentes actividades acaecidas en el cuerpo humano. En función del origen de estas señales reciben los siguientes nombres: electrocardiograma (ECG) para las señales originadas por las contracciones del corazón; electromiograma (EMG) para las generadas en los músculos; electroencefalograma (EEG) para aquellas generadas en el cerebro, etc.



Figura 1.2: Ejemplo de ECG

1.1. Alcance y estructura del proyecto

El **objetivo de este proyecto** es realizar un sistema capaz de captar señales de electroencefalogramas (EEG) manteniendo una buena **relación prestaciones/coste**. El sistema estará compuesto de dos tarjetas, una de acondicionamiento y de adquisición de datos basada en el circuito integrado ADS1299 y otra de **procesamiento y transmisión** de dicha información. Esta última es el objetivo del presente proyecto. La plataforma de procesamiento estará basada en un procesador de altas prestaciones, dispondrá de interfaces **Wifi, Bluetooth y almacenamiento USB** para la transmisión y almacenamiento de los datos respectivamente.

En la primera fase del proyecto se seleccionará el microcontrolador más adecuado entre los existentes en el mercado analizando características como: capacidad de procesado, interoperación con otros dispositivos, prestaciones...

Se compararán los microcontroladores ofrecidos por los distintos fabricantes (ST Microelectronics, Texas instruments, etc) y finalmente, se escogerá aquel que mejor se adapte a las necesidades del proyecto siendo los principales candidatos los de la familia **ARM-M4 STM32F4x** por su excelente relación prestaciones-coste.

Se valorará también la posibilidad de utilizar diferentes herramientas para la programación del microcontrolador y las alternativas *Open Source* en caso de existir.

Una vez completado el diseño eléctrico de la tarjeta se procederá al **diseño de la PCB equivalente**, la cual se implementará utilizando **tecnología SMT** en su mayor parte. Para el diseño de la placa se utilizará KiCad por las numerosas ventajas que presenta al ser software libre y la gran cantidad de información que se puede encontrar sobre el funcionamiento del mismo. Tras depurar la PCB, se implementará un sencillo **firmware** que permita testear el hardware diseñado y hacer una adquisición básica utilizando la tarjeta desarrollada junto con la de adquisición cedida por parte de Nerea Urrestarazu.

Adicionalmente se pondrá en marcha un **programa para el ordenador** basado en LabView donde presentar los datos recibidos.

1.2. Base teórica

Antes de empezar a desarrollar el dispositivo ya mencionado es indispensable realizar una investigación previa del origen de las señales que se quieren adquirir, haciendo especial hincapié en las características de las mismas¹.

En el cerebro se realizan **millones de conexiones** diarias entre las células que la conforman. Dichas células reciben el nombre de **neuronas**. La comunicación entre ellas se realiza mediante el **intercambio de neurotransmisores**, proceso que recibe el nombre de sinapsis y que provoca una corriente eléctrica de mayor o menor intensidad que permite la inhibición o activación de las distintas neuronas.

La suma de todas las corrientes eléctricas conforma lo que se denomina **actividad bioeléctrica cerebral**. Mediante el uso de sensores es posible medirla y representarla.

Las señales del EEG dependen del estado de conciencia del usuario y se pueden clasificar en función de su amplitud y frecuencia. Las principales ondas son las siguientes:

- **Ondas delta** (hasta 3.5 Hz). Término introducido por W. G. Walter en 1937 para describir a estas ondas de baja frecuencia y alta intensidad (unos centenares de μV). Tienen lugar en niños de corta edad y en adultos sólo en estado de sueño profundo, inconsciencia o situaciones que aumenten la presión intercraneal como tumores cerebrales.

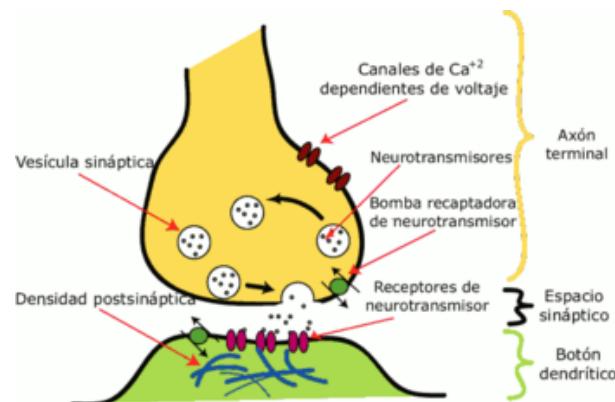


Figura 1.3: Sinapsis neuronal [2]

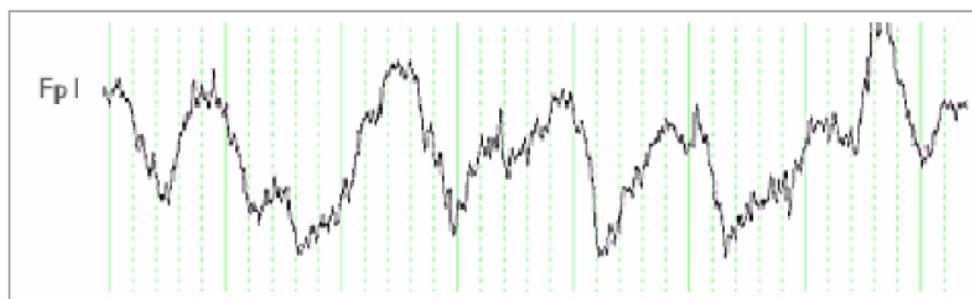


Figura 1.4: Ondas delta [1]

¹El contenido de esta sección se ha obtenido en su mayor parte de las diapositivas de la asignatura de Señales e Imágenes Médicas impartida por Jose Javier Serrano Olmedo [1]

- **Ondas theta** (3.5 - 7.5 Hz). Término también usado por Walter aunque mucho más tarde, en 1953. Estas ondas de amplitud inferior a $20 \mu\text{V}$ se dan durante el proceso de maduración en toda la corteza cerebral, aunque predominan en la región occipital y temporal y es más rápida en la zona frontal. Dominante en niños entre 5 y 7 años y aún quedan rastros en la juventud. En adultos y adolescentes se asocia a situaciones emocionales y pensamientos de tipo creativo, a estrés o a desordenes psíquicos.

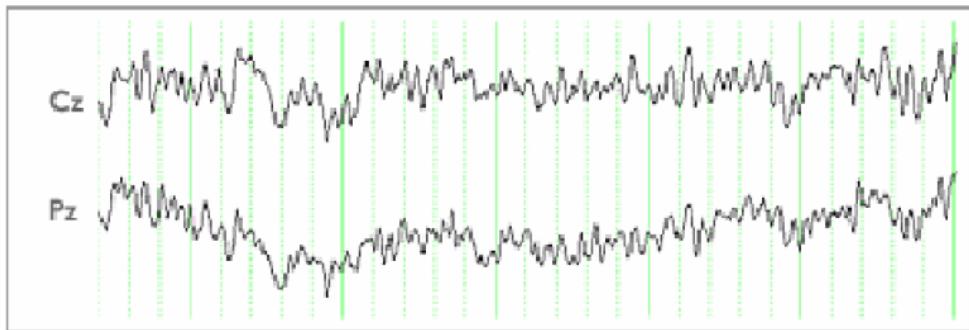


Figura 1.5: Ondas theta [1]

- **Ondas alfa** (7.5 - 12.5 Hz). Berger utilizó el término de ritmo alfa para ráfagas de 20-100 μV de amplitud y gran periodicidad a esas frecuencias predominantes sobre la región occipital pero que aparecen en todo el córtex. Se asocian a estados de relajación, de inactividad y son muy patentes en ausencia de estímulos visuales. Existe mucha variabilidad interpersonal en el ritmo alfa.

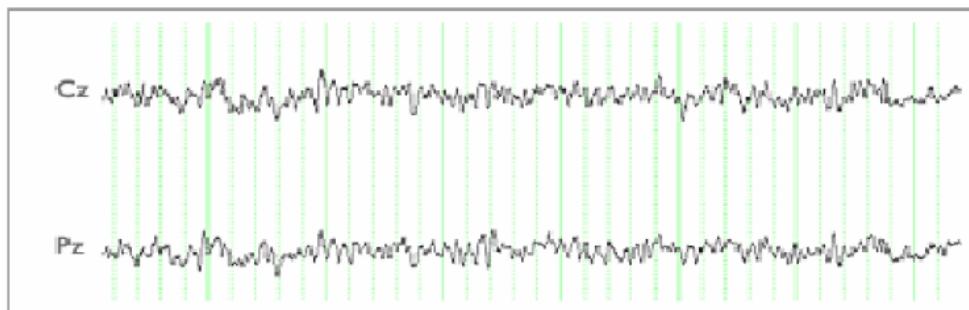


Figura 1.6: Ondas alfa [1]

- **Ondas beta** (12.5 - 30 Hz). Estas señales de pequeña amplitud, por debajo de $20\mu\text{V}$, son bastante comunes y predominan durante la edad adulta. Suele dividirse en beta baja, beta media y beta alta. El ritmo beta bajo se suele localizar en los lóbulos frontal y occipital y los otros dos están menos localizados. Más irregular que el ritmo alfa, se asocia a actividad psicofísica, estados de agitación, alerta o la actividad mental que se realiza en la resolución de problemas.

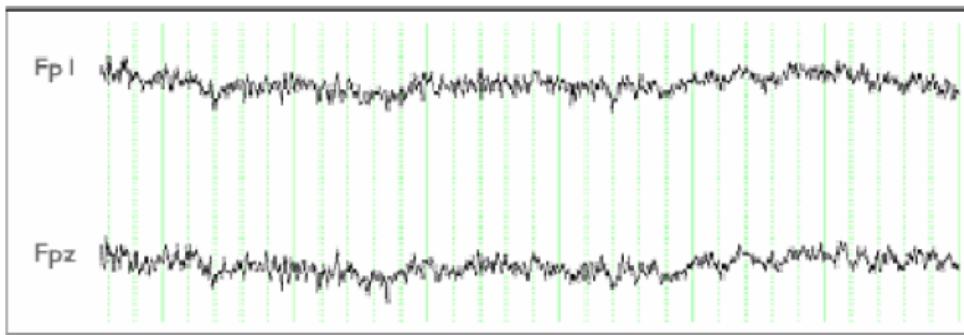


Figura 1.7: Ondas beta [1]

- **Ondas gamma** (12.5 - 30 Hz). Esta banda tiene un pico de resonancia, similar a los ritmos alfa, cercano a 40Hz al que se da el nombre de epsilon que se asocia a actividad mental abstracta que interviene en percibir como único por ejemplo el olor, los rasgos de la cara, la personalidad y la voz de una persona. Actualmente hay quien habla de ondas de frecuencias superiores a 50 Hz, son las hipergamma hasta 100 Hz y las lambda hasta 200 Hz, se dice aparecen en los monjes tibetanos en estados de meditación profunda

Para la realización de un electroencefalograma existe una disposición estándar en la colocación de los sensores denominada **sistema 10-20**. Se trata de un sistema internacional que indica la posición en la que se deben colocar los electrodos para una medición del EEG. Este sistema separa el cuero cabelludo en torno a un eje Z, colocando los electrodos numerado de manera par a la derecha y los impares a la izquierda (tal y como se puede ver en la figura 1.8).

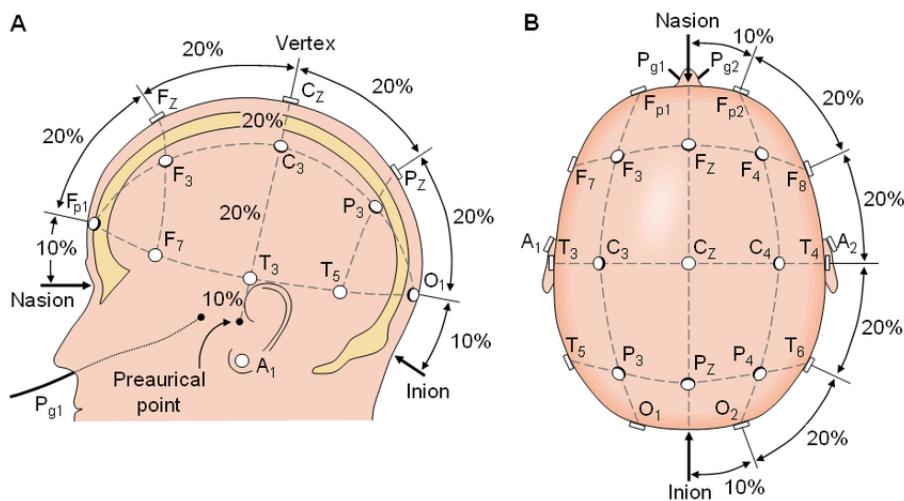


Figura 1.8: Esquema de colocación de los electrodos [1]

1.3. Tipos de electrodos

Los electrodos hacen la función de interfaz adaptadora entre los distintos medios por los que se transmiten las señales permitiendo así obtener una señal de mejores características. Este es un funcionamiento similar al de los huesos del oído, pues estos adaptan la señal acústica para que la señal percibida en la coclea tenga unas características determinadas. El funcionamiento de la mayoría de los dispositivos de adquisición de señales en el cuerpo humano depende de estos elementos y hacen uso de dos tipos principales, de trabajo y auxiliar.

El electrodo denominado “de trabajo” (*working electrode*) es el que adquiere la señal de interés. El otro, denominado auxiliar, es el encargado de crear una referencia. El esquema de la figura 1.9 muestra un esquema básico de funcionamiento:

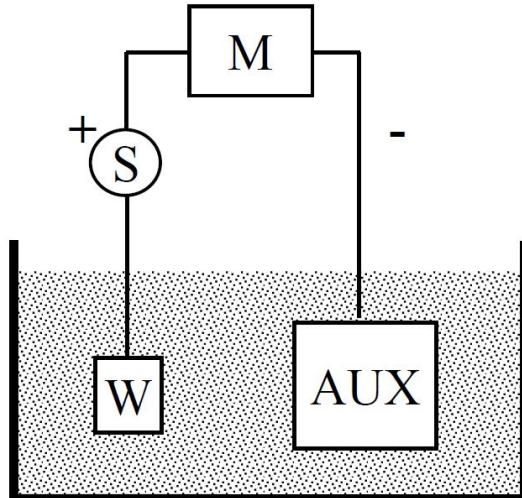


Figura 1.9: Electrodos de en un sistema de adquisición [1]

En función del método utilizado por el electrodo para hacer la interfaz con el cuerpo humano estos electrodos se pueden dividir en dos grandes grupos, cada uno con sus ventajas e inconvenientes.

1.3.1. Electrodos húmedos

Los electrodos húmedos se caracterizan por utilizar algún material acuoso para realizar la interfaz entre el dispositivo y el cuerpo humano. Al estar conformados por varios materiales de distinta naturaleza estos electrodos suelen ser de construcción compleja. La figura 1.10 muestra el esquema de un electrodo húmedo.

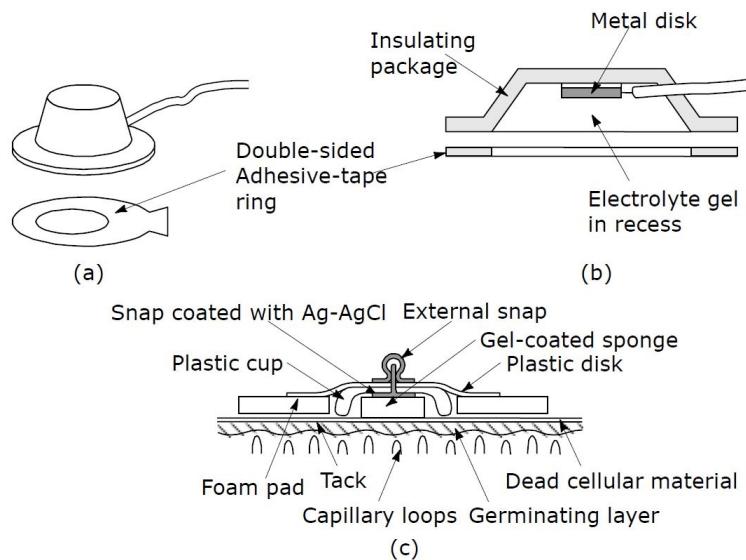


Figura 1.10: Esquema de un electrodo húmedo [1]

Estos dispositivos suelen dar muy buen resultado, consiguiendo señales con una buena amplitud y SNR pero el material utilizado como interfaz suele ser desechable lo que obliga a usar electrodos de usar y tirar o a tener que dedicar mucho tiempo en la preparación y rellenado con gel de la cámara interior del dispositivo. Además, desde el punto de vista de la comodidad del usuario, la utilización de electrodos húmedos puede resultar incómoda al obligarlo a lavar la zona antes y después de su utilización.

La figura 1.11 muestra un ejemplo de adquisición de EEG utilizando electrodos húmedos.

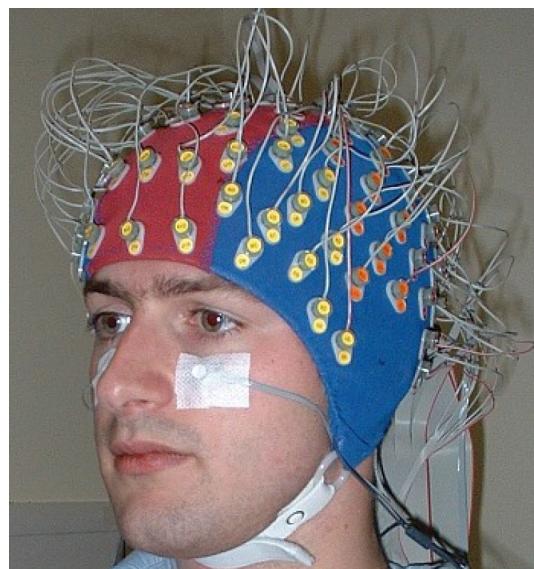


Figura 1.11: Medida de un EEG usando electrodos húmedos [1]

1.3.2. Electrodos secos

Los electrodos secos utilizan materiales sólidos cuya interacción con el cuerpo humano resulta favorable. Al realizar una interfaz con materiales sólidos las señales se transmiten con mayor dificultad, pero este sistema resulta mucho más cómodo para el usuario y los electrodos son de fabricación más sencilla y reutilizables. La figura 1.12 muestra una representación 3D de un electrodo seco (derecha) y un electrodo seco real junto con sus correspondientes conectores (izquierda).



Figura 1.12: Electrodo seco real(izquierda) y representación 3D (derecha) [1]

Por comodidad durante la realización de este proyecto se utilizarán principalmente electrodos secos pero la posibilidad de usar electrodos húmedos se mantendrá, pues la utilización de unos u otros afecta al usuario final y a los resultados pero apenas al equipo encargado de su adquisición.

2

Estado del Arte

Con el paso del tiempo se ha demostrado que una de nuestras mayores virtudes como seres humanos es la habilidad de **aprovechar el saber cultivado por otras personas** para realizar nuevos descubrimientos con mayor facilidad. En la actualidad, con la ayuda de Internet esta ventaja se ha visto potenciada hasta límites insospechados.

Como se ha mencionado con anterioridad en el capítulo 1, a lo largo de los años se han desarrollado numerosas alternativas a los dispositivos presentes en los hospitales y laboratorios utilizados normalmente para el estudio del cerebro.

Aunque se han invertido muchos recursos en estos dispositivos, el objetivo es permitir **ampliar el número de personas** capaces de estudiar el cerebro humano, consiguiendo así aumentar las posibilidades de mejorar nuestro conocimiento sobre el mismo.

De esta forma debería ser más fácil realizar nuevos descubrimientos como, por ejemplo, nuevas formas de diagnosticar enfermedades o de realizar una comunicación hombre-máquina para aquellas personas que por algún tipo de discapacidad física o mental no puedan utilizar los medios convencionales.

2.1. Dispositivos similares

A lo largo de este **capítulo** se presentarán algunos de los **dispositivos capaces de capturar un EEG**, a nivel personal, enfocados a la docencia o, por supuesto, diseñados con el fin de realizar un producto final que vender a terceros.

2.1.1. Proyectos personales

En internet se pueden encontrar algunos ejemplos de personas que han dedicado su tiempo a crear dispositivos capaces de captar un EEG.

Instructables

La página web **instructables.com** ha propuesto a sus usuarios la creación de un sistema de adquisición de EEG y ECG sencillo y barato. El dispositivo final resulta muy interesante, pues con unas pocas resistencias, condensadores y un par de amplificadores operacionales son capaces de **montar un dispositivo funcional**. La figura 2.1 muestra el esquemático final del dispositivo.

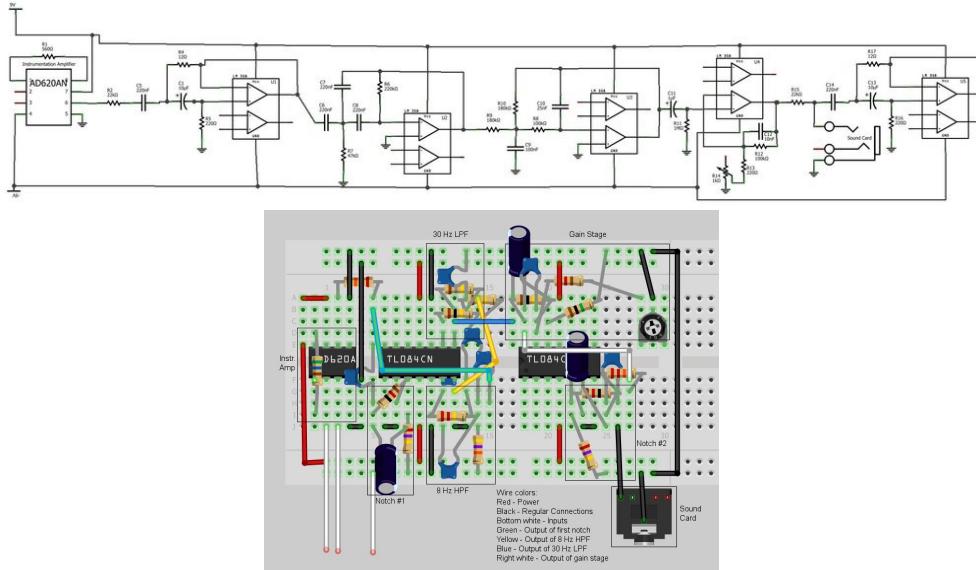


Figura 2.1: Esquema electrónico casero para la captura de un EEG [3]

Este sistema es muy interesante y para fines educativos cumple perfectamente su función pero **no tiene en cuenta el ruido que afecta a la señal** ni otras características importantes como son el **aislamiento** del paciente de la red eléctrica.

En su página web no especifican el coste total del proyecto pero en base al número de componentes se estima que el **coste final es superior a los 60€**.

OpenEEG

OpenEEG es un proyecto de software libre que pone a disposición de cualquiera que lo necesite un conjunto de herramientas, software y esquemáticos que permiten crear un sistema de adquisición EEG utilizando una base que **se encuentra en continuo desarrollo**.

A lo largo de Internet se pueden encontrar un gran número de pruebas de concepto y prototipos basados en este proyecto. En general muestran los resultados obtenidos pero hay algunos que además guían a los más curiosos que se atrevan a intentar construirse uno. Este es el caso de la web <http://openeeg.sourceforge.net/buildeeg/> (aún en construcción), donde un usuario anónimo muestra sus resultados y anima a cualquiera con conocimientos básicos de electrónica a intentar repetir el proceso, proporcionando esquemáticos, consejos y software.

El resultado final de este proyecto es el que se muestra en la figura 2.2.

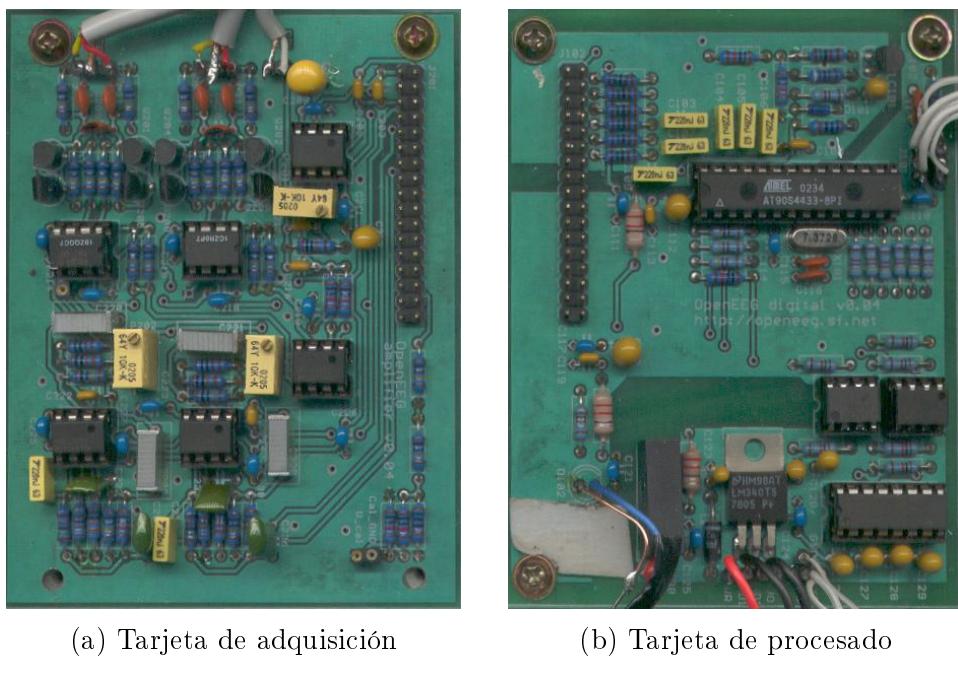


Figura 2.2: Elementos que conforman OpenEEG [4]

El sistema está dividido en **dos placas** principales, una de **adquisición** encargada del acondicionamiento de la señal mediante resistencias, condensadores y amplificadores operacionales y otra de **procesado** donde el núcleo es un microcontrolador de ATMEL (presentes también en Arduino).

Resulta evidente que este proyecto presenta un **acabado más profesional** ya que tiene en cuenta factores muy importantes como el acondicionamiento de la señal para minimizar el ruido y el aislamiento de la alimentación del paciente.

El **precio total** estimado de este sistema es de **180€** aproximadamente.

Beanie

Esta curiosa iniciativa utiliza un analizador de EEG acoplado a un gorro para que este se ilumine en función del estado y concentración de la persona.



Figura 2.3: Beanie en funcionamiento [5]

El funcionamiento es muy similar al resto de dispositivos ya explicados: se adquieren señales desde electrodos situados en la cabeza, se filtra la señal para eliminar el ruido y se presenta al usuario.

Para la adquisición de la señal utiliza un chip de la compañía Neurosky llamado **ThinkGear ASIC Module** que tiene integrado un sistema de adquisición y filtrado de señales. Posteriormente está se transfiere a un módulo **TinyLily de Arduino** que se encargará de realizar un segundo procesado para iluminar el pompón del gorro de la forma deseada.

El dispositivo se monta con relativa facilidad (1-2 días) y su **precio** ronda los **130€**.

GitHub

Por último es importante destacar la presencia de GitHub. Esta plataforma permite que la gente suba una gran cantidad de proyectos que normalmente están accesibles.

Aunque los proyectos encontrados no suelen contener información acerca del diseño de placas y el precio final, sí que incluyen una gran cantidad de código que será utilizado como referencia para la implementación del código de este proyecto. Se incluye en la bibliografía algunas de los repositorios de mayor interés.

2.1.2. Proyectos docentes

Los proyectos llevados a cabo por universidades y otras instituciones centradas en el estudio y la docencia suelen ser de difícil acceso, pues la documentación generada tras un proyecto se suele aprovechar para realizar publicaciones en las revistas científicas más conocidas.

Sin embargo, en ocasiones los Trabajos Fin de Carrera abordan temas similares y estos si que se encuentran disponibles para cualquiera que tenga acceso a Internet.

Un ejemplo de esta situación es el Trabajo Fin de Máster de la alumna Nerea Urrestarazu. En este se aborda la creación de un sistema de adquisición de EEG haciendo uso de un convertidor analógico-digital y un ordenador. Este proyecto se ha utilizado como base para construir otro que mejore sus prestaciones, así que sus características serán ampliadas en posteriores capítulos.

El coste final del proyecto contando sólo los materiales asciende a 100€ aproximadamente.

2.1.3. Sistemas comerciales

Muchas empresas han visto este campo como una **oportunidad de negocio** y han desarrollado varias aplicaciones que hacen uso de la medida de EEG para diversos propósitos.

The XWave Headset

Este sistema es exclusivo de usuarios de iPhone. Por el momento incluye una diadema que, mediante el contacto con la frente, permite ver al usuario sus ondas cerebrales y **controlar ciertas aplicaciones**.



Figura 2.4: Imagen promocionando el producto

El estudio que realiza del EEG no es muy exhaustivo de modo que no es un candidato a utilizar para estudios médicos pero sí que permite a los usuarios familiarizarse con la terminología y acostumbrarse a dispositivos similares.

Su precio es de **99.90\$**, (86€).

NeuroSky

Esta compañía ha creado una diadema similar a la presentada con anterioridad. Esta compañía pone a disposición del usuario un paquete de inicio que incluye la diadema, software y una guía de inicio. Este software contiene entre otras aplicaciones una **película cuya trama cambia en función del estado mental** del usuario.

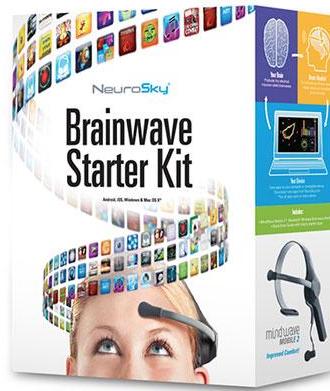


Figura 2.5: CAPTION

El precio comercial es de 99.99\$ (86€).

OpenBCI

Los ejemplos anteriores tienen características muy interesantes pero ambos son poco adecuados para realizar estudios médicos de las ondas cerebrales. Siguiendo esta premisa se ha creado un proyecto denominado OpenBCI con una extensa comunidad de desarrolladores.

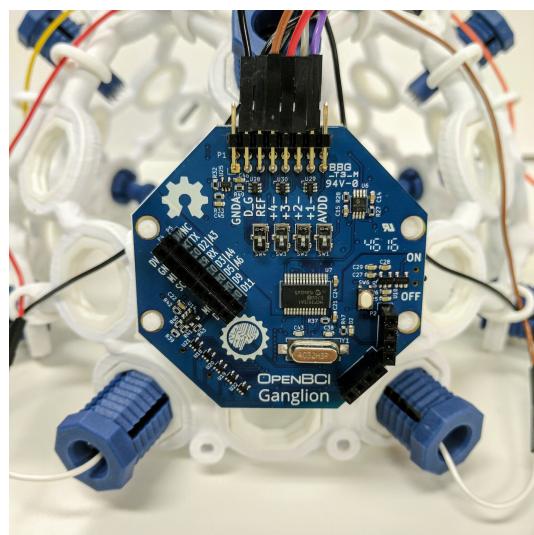


Figura 2.6: Módulo Ganglion montado sobre un casco [6]

Este sistema hace uso del **convertidor analógico ADS1299** para realizar una adquisición de las señales procedentes del cerebro usando **4 canales**. A continuación realiza un procesado inicial y las transmite a un ordenador, donde serán filtradas y reprocesadas para extraer la información de interés.

Está basado en la utilización de módulos, siendo **necesario comprar el módulo base (Ganglion) por 200\$ (175€)** para funcionar mientras que el resto de módulos son opcionales y son usados para aumentar sus funcionalidades.

2.2. Comparativa

Las mayoría de las alternativas comerciales presentadas a lo largo de este capítulo intentan maximizar el confort del usuario aunque esto afecte al rendimiento del dispositivo.

Los sistemas orientados a la investigación se centran en los resultados mientras dejan el confort en segundo plano. Por supuesto intentan que el dispositivo sea lo más cómodo posible pero no permiten que esto afecte a las señales medidas.

La tabla 2.1 hace una comparativa de los equipos anteriormente expuestos

Dispositivo	Apto para la investigación / Aplicable a la medicina	Precio[€]
Instructables	Sí ¹	60
OpenEEG	Sí	180
Beanie	No	130
XWave Headset	No	86
NeuroSky	No	86
OpenBCI	Sí	175

Tabla 2.1: Comparativa entre los dispositivos

Todos los dispositivos anteriores tienen un precio que ronda los cientos de euros. En función del objetivo del equipo algunos sacrifican parte de las funcionalidades para que el coste disminuya pero aun así el resultado final puede significar un esfuerzo económico para el usuario medio.

¹Es posible aplicarlo a esos campos pero no se debe olvidar que no realiza una eliminación del ruido

3

Diseño

Este proyecto consiste en una **ampliación y cambio de enfoque** de otro proyecto llevado a cabo de manera simultánea por una alumna de la Universidad Politécnica de Madrid llamada Nerea Urrestarazu que, a su vez, se basa en el Kit de demostración de rendimiento del ADS1299 proporcionado por Texas Instrument.



Figura 3.1: Placa final del proyecto base

3.1. Diseño base

El **proyecto original** consiste en el **diseño y desarrollo** de una **placa de adquisición de EEG** haciendo uso de los integrados ADS1299 junto con un sistema de transmisión hacia el ordenador tanto inalámbricamente como a través de USB. La figura 3.2 muestra las partes que componen dicho diseño.

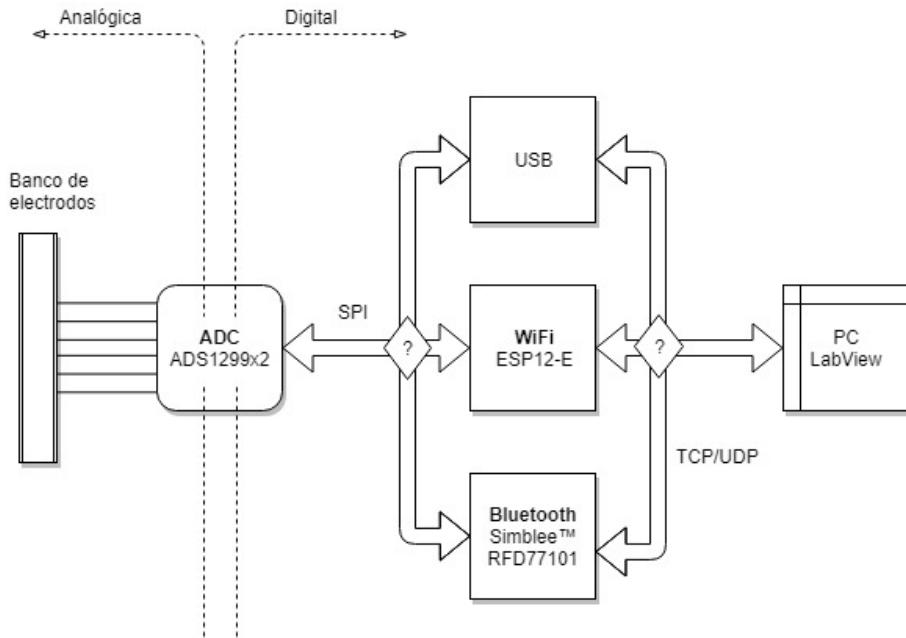


Figura 3.2: Esquema del proyecto base

3.1.1. Adquisición de datos

La parte encargada de la **adquisición** está compuesta por un par de **bancos de electrodos** dispuestos en los laterales de la placa seguidos por un filtro paso-bajo con frecuencia de corte de 6.79kHz, encargado de eliminar las componentes de frecuencias muy altas, no deseadas en el estudio de un EEG. A continuación se encuentran conectados a sus respectivos bancos los **ADC ADS1299**.

Estos convertidores son capaces de adquirir información de forma independiente o en modo “*Daisy Chain*” y **transmitirla a través de SPI** hacia otros dispositivos cuya misión será gestionarla.



Figura 3.3: Convertidor Analógico-Digital ADS1299

El SPI presente en el convertidor permite leer todos los registros del ADS y escribir la gran mayoría. Aunque normalmente se leen los relacionados con los datos convertidos, también es posible saber el estado de los GPIO o el identificador único del dispositivo leyendo su registro asociado.

La **configuración** del ADS se realiza mediante la **escritura de ciertos registros**, cada uno asociado a un parámetro específico. La tabla 3.1 muestra los registros disponibles, tanto de lectura como de configuración, una descripción básica y la dirección de memoria asociada a los mismos.

ADDRESS	REGISTER	DEFAULT SETTING	REGISTER BITS									
			7	6	5	4	3	2	1	0		
Read Only ID Registers												
00h	ID	xxh	REV_ID[2:0]			1	DEV_ID[1:0]		NU_CH[1:0]			
Global Settings Across Channels												
01h	CONFIG1	96h	1	DAISY_EN	CLK_EN	1	0	DR[2:0]				
02h	CONFIG2	C0h	1	1	0	INT_CAL	0	CAL_AMPO	CAL_FREQ[1:0]			
03h	CONFIG3	60h	PD_REFBUF	1	1	BIAS_MEAS	BIASREF_INT	PD_BIAS	BIAS_LOFF - SENS	BIAS_STAT		
04h	LOFF	00h	COMP_TH[2:0]			0	ILEAD_OFF[1:0]		FLEAD_OFF[1:0]			
Channel-Specific Settings												
05h	CH1SET	61h	PD1	GAIN1[2:0]			SRB2	MUX1[2:0]				
06h	CH2SET	61h	PD2	GAIN2[2:0]			SRB2	MUX2[2:0]				
07h	CH3SET	61h	PD3	GAIN3[2:0]			SRB2	MUX3[2:0]				
08h	CH4SET	61h	PD4	GAIN4[2:0]			SRB2	MUX4[2:0]				
09h	CH5SET ⁽¹⁾	61h	PD5	GAIN5[2:0]			SRB2	MUX5[2:0]				
0Ah	CH6SET ⁽¹⁾	61h	PD6	GAIN6[2:0]			SRB2	MUX6[2:0]				
0Bh	CH7SET ⁽²⁾	61h	PD7	GAIN7[2:0]			SRB2	MUX7[2:0]				
0Ch	CH8SET ⁽²⁾	61h	PD8	GAIN8[2:0]			SRB2	MUX8[2:0]				
0Dh	BIAS_SENSP	00h	BIASP8 ⁽²⁾	BIASP7 ⁽²⁾	BIASP6 ⁽¹⁾	BIASP5 ⁽¹⁾	BIASP4	BIASP3	BIASP2	BIASP1		
0Eh	BIAS_SENSN	00h	BIASN8 ⁽²⁾	BIASN7 ⁽²⁾	BIASN6 ⁽¹⁾	BIASN5 ⁽¹⁾	BIASN4	BIASN3	BIASN2	BIASN1		
0Fh	LOFF_SENSP	00h	LOFFP8 ⁽²⁾	LOFFP7 ⁽²⁾	LOFFP6 ⁽¹⁾	LOFFP5 ⁽¹⁾	LOFFP4	LOFFP3	LOFFP2	LOFFP1		
10h	LOFF_SENSN	00h	LOFFM8 ⁽²⁾	LOFFM7 ⁽²⁾	LOFFM6 ⁽¹⁾	LOFFM5 ⁽¹⁾	LOFFM4	LOFFM3	LOFFM2	LOFFM1		
11h	LOFF_FLIP	00h	LOFF_FLIP8 ⁽²⁾	LOFF_FLIP7 ⁽²⁾	LOFF_FLIP6 ⁽¹⁾	LOFF_FLIP5 ⁽¹⁾	LOFF_FLIP4	LOFF_FLIP3	LOFF_FLIP2	LOFF_FLIP1		
Lead-Off Status Registers (Read-Only Registers)												
12h	LOFF_STATP	00h	IN8P_OFF	IN7P_OFF	IN6P_OFF	IN5P_OFF	IN4P_OFF	IN3P_OFF	IN2P_OFF	IN1P_OFF		
13h	LOFF_STATN	00h	IN8M_OFF	IN7M_OFF	IN6M_OFF	IN5M_OFF	IN4M_OFF	IN3M_OFF	IN2M_OFF	IN1M_OFF		
GPIO and OTHER Registers												
14h	GPIO	0Fh	GPIOD[4:1]				GPIOC[4:1]					
15h	MISC1	00h	0	0	SRB1	0	0	0	0	0		
16h	MISC2	00h	0	0	0	0	0	0	0	0		
17h	CONFIG4	00h	0	0	0	0	SINGLE_SHOT	0	PD_LOFF_COMP	0		

Tabla 3.1: Tabla de registros de la familia ADS [7]

Una descripción más detallada de cada uno de los bits de cada registro se puede encontrar en el *datasheet* del componente [7].

Como se puede ver en la tabla 3.1, los convertidores cuentan con una gran cantidad de opciones de configuración. Para el desarrollo de este proyecto **se ha implementado un sistema de configuración que permite la lectura y escritura de todos los registros**.

3.1.2. Transmisión de datos

Tras completar el proceso de adquisición de datos resulta necesario transmitir dicha información hacia un dispositivo capaz de procesarla. Para ello la placa original contaba con dos alternativas. La primera consiste en, mediante **USB** y acopladores aislantes, transmitir la información a un ordenador.

La segunda hace uso de dos tecnologías inalámbricas distintas que funcionan de forma

excluyente y son seleccionables con un *jumper*: **WiFi** o **Bluetooth**.

WiFi

Para la transmisión de datos a través de **WiFi** se seleccionó el **módulo ESP12-E**, basado en el SoC ESP2866, también conocido nodemcu.

Este cuenta con un microcontrolador (MCU) embebido de 32 bits (Tensilica L106) con una memoria RAM de 36kB y una velocidad de reloj de la CPU de hasta 80MHz, proporcionando suficiente potencia para las tareas básicas.

Así mismo se incluye montado en el mismo paquete una memoria flash de 4MB en la que almacenar el código de los programas que se ejecutarán y una antena embebida, dotando al módulo de conectividad en la banda de 2.4GHz.



Figura 3.4: ESP8266

A efectos de diseño es muy importante saber cuales serán las entradas/salidas del dispositivo así como los **pines dedicados para su programación**. La figura 3.5 muestra un resumen de todas las funciones de cada uno de los pines. Como se puede observar, el SPI hace uso de los pines 5, 6, 7 y 16. Por otro lado el UART, necesario para la programación del micro hace uso de los pines 16 y 17. Dichos pines deberán **reservarse** posteriormente en la fase de diseño de la PCB.

El dispositivo tiene tres modos de arranque dependiendo del sitio desde el que cargue el código y la selección de uno u otro modo viene determinada por los pines MTDO, GPIO0 y GPIO2.

La tabla 3.2 los distintos modos de arranque y el estado en el que deben estar de los pines para iniciar en dicho modo.

MTDO	GPIO0	GPIO2	Modo	Descripción
L	L	H	UART	Descarga el código desde UART
L	H	H	Flash	Carga desde memoria Flash a través de SPI
H	x	x	SDIO	Carga desde una tarjeta SD

Tabla 3.2: Modos de arranque del ESP12-E [8]

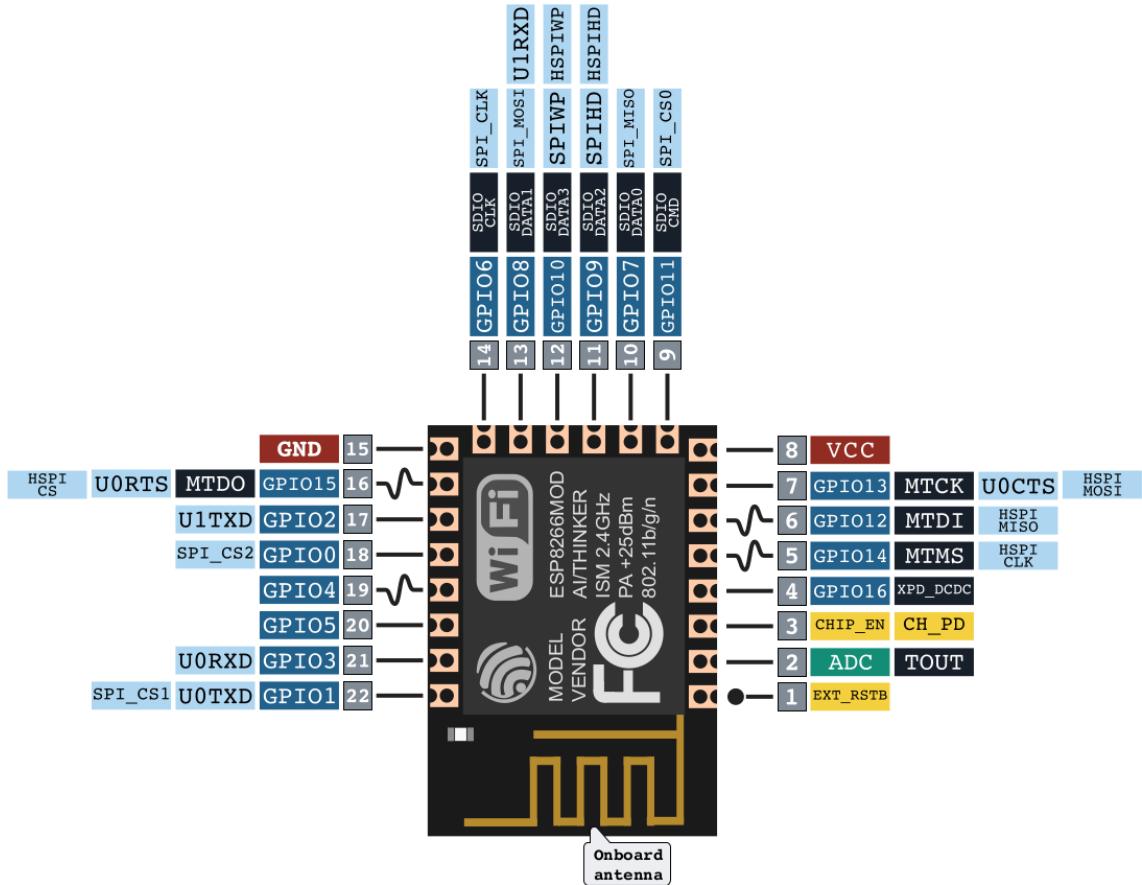


Figura 3.5: Resumen de todas las Entradas/Salidas del ESP12-E [9]

Bluetooth

Para la transmisión de datos a través de **Bluetooth** se seleccionó el módulo Simblee™ RFD77101 ya que al igual que el módulo WiFi, cuenta con interfaces de **comunicación a través de SPI** para conectarse con el ADC (pines 21, 22, 31 y 32) y de UART para su programación posterior programación (pines 23 y 24).



Figura 3.6: Simblee™ RFD77101 [10]

El módulo presenta un ARM Cortex-M0 como CPU con 128KB de memoria Flash, 24KB de RAM y una frecuencia de reloj de 16MHz.

Todos los dispositivos de transmisión inalámbrica anteriormente mencionados permiten su **programación utilizando el IDE de Arduino** lo cual facilita sensiblemente el proceso de desarrollo y prototipado teniendo la ventaja adicional de que es Software Libre.

USB

Como este proyecto tiene como objetivo independizar el sistema lo máximo posible del ordenador se ha optado por **desestimar** el sistema de transmisión por USB conservando solamente la interfaz inalámbrica.

3.2. Diseño final

Llegados a este punto se han analizado las características más importantes de cada uno de los elementos presentes en el sistema original, siendo los más importantes las distintas interfaces de comunicación y los pines de programación.

Con esta información ya es posible independizar cada uno de esos elementos y crear un **nuevo diseño que cumpla con las especificaciones de este proyecto**.

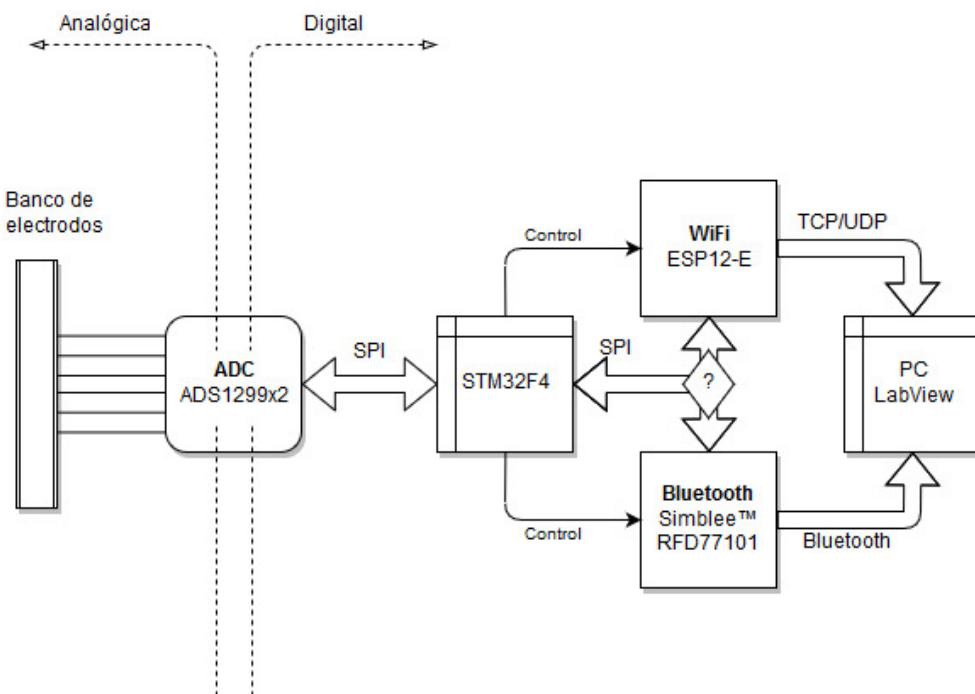


Figura 3.7: Esquema del proyecto final

El nuevo sistema estará compuesto por **tres partes**:

- En primer lugar etapa de **adquisición** compuesta de un **banco de electrodos** con sus correspondientes filtros analógicos y **dos ADC**.
- Posteriormente un **microcontrolador** se encargará de realizar el **procesado** de la señal y la **gestión** de los distintos elementos.
- Por último la transmisión de datos se realizará de forma **inalámbrica** a un ordenador u otro dispositivo mediante **Bluetooth o WiFi**.

Como se puede observar en la figura 3.7, el ordenador sigue estando presente en el sistema pero en esta ocasión su función se limita a mostrar la información siendo fácilmente sustituible en un futuro por un dispositivo menos potente y barato. Al utilizar un microcontrolador en la propia placa de adquisición se consigue aumentar la independencia del sistema y se dota de unas características muy interesantes, tanto de procesado de señal como de almacenaje de la misma o gestión del consumo.

3.2.1. Selección del microcontrolador

El microcontrolador es el núcleo del sistema. Para poder interactuar con todos los elementos anteriormente descritos deberá contar con las siguientes características:

- Bajo precio.
- Bajo consumo.
- Capacidad procesado de señal.
- Para poder utilizar el máximo de la velocidad de adquisición de los ADS (16kSPS) y evitar cuellos de botella, deberá contar con un Bus SPI de al menos 5.85Mb/s dedicados para la transmisión y la misma cantidad para recepción o dos buses.

$$\text{Tasa de transferencia} = 16k \text{ Muestras/s} * 8 \text{ canales} * 24\text{bits} * 2 \text{ ADS} = 5,85\text{Mb/s}$$

Hay una gran cantidad de microcontroladores en el mercado que podrían utilizarse para este proyecto pero, de entre todos los disponibles, aquellos con arquitectura **ARM** son los que mejor se adaptan a las especificaciones. Concretamente los más adecuados son aquellos englobados en la familia **Cortex M4** ya que están especialmente diseñados con DSP integrado para conseguir un **rendimiento máximo minimizando el consumo**.



(a) Logo de Texas Instrument [11]



(b) Logo de STMicroelectronics[12]

Figura 3.8: Principales fabricantes contemplados

CAPÍTULO 3. DISEÑO

La tabla 3.3 muestra varios dispositivos de esa familia vendidos por Texas Instrument o STMicroelectronics y un resumen de sus características principales.

	Texas Instruments	STMicroelectronics		
	TM4C123GH6PM	TM4C1294NCPDT	STM32F405	STM32F469
FCPU [MHz]	80	120	168	180
Flash [kB]	256	1024	1024	2048
RAM [kB]	32	256	192	384
SPI	x4	x4	x2 + 1	x6
Precio [€]	8,72	12,74	9,24	13,48

Tabla 3.3: Comparativa entre distintos MCUs

Todos los dispositivos anteriormente contemplados tienen DSP integrados así como una Unidad de Punto Flotante (FPU) que permite realizar **operaciones matemáticas avanzadas** de forma óptima.

Entre los dispositivos anteriores, los pertenecientes a la familia **STM** presentan mejor relación coste/prestaciones, pero el verdadero factor diferenciador son las herramientas dispuestas para la comunidad por parte del fabricante.

En la página web se pueden encontrar distintas **aplicaciones** y **documentación** que facilitan sensiblemente el proceso de desarrollo para esta plataforma.

Finalmente se seleccionó el **MCU STM32F405** en el formato **LQFP64** ya que sus características se ajustan perfectamente a las especificaciones, manteniendo unas muy buenas prestaciones y un precio bastante bajo. Todo esto con el valor añadido de que ya se contaba con la placa de desarrollo **STM32F4 Discovery** lo cual permitió comenzar con el aprendizaje y estudio del entorno sin la necesidad de esperar al diseño, impresión y soldado de la placa final.



Figura 3.9: Placa de desarrollo STM32F4 Discovery

El MCU cuenta con 3 buses SPI independientes que pueden funcionar en modo *Full Duplex*. El SPI₁ es capaz de funcionar hasta 42Mb/s mientras que los SPI₂ y SPI₃ pueden comunicar información hasta 21Mb/s. Los pines dedicados a dichos buses se pueden consultar en el *Datasheet* del componente [13].

Otra característica interesante de este MCU es la presencia de forma nativa de un gestor de USB On The Go (OTG) permitiendo así conectar un dispositivo USB para almacenar información a largo plazo.

3.2.2. Alimentación

Tras seleccionar todos los elementos se va a proceder a escoger una alimentación que permita a todos los dispositivos funcionar en condiciones óptimas.

La tabla 3.4 muestra un resumen de todos los elementos presentes en el sistema junto con los rangos de voltaje recomendados por los fabricantes para su alimentación.

Dispositivo	V _{min} [V]	V _{max} [V]
Simblee BT	1.8	3.6
ESP12-E	3.0	3.6
STM	1.8	3.6
ADS (Digital)	1.8	3.6
ADS (Analógica)	4.75	5.25
USB	5	5

Tabla 3.4: Rangos de alimentación de todos los elementos

De la tabla 3.4 se deduce que para que todos los dispositivos funcionen correctamente **será necesario dotar a la placa de 5 voltios** con los que se podrá alimentar la parte analógica del ADS así como el USB mientras que la parte digital se puede alimentar en el rango de 3V a 3.6V.

Por motivos de compatibilidad con el diseño anterior y tras comprobar que se cumple con los requisitos impuestos por los nuevos elementos del sistema se ha optado por mantener el mismo esquema de alimentación que en el proyecto base.

El sistema de alimentación se compone de dos partes principales, cada una encargada de proporcionar el voltaje deseado manteniendo el ruido generado por el mismo al mínimo.

Alimentación 3.3V

Para conseguir un voltaje de **3.3V** estable se ha utilizado el regulador **AZ1117C-3.3** ya que es capaz de conseguir una precisión para el voltaje de salida del $\pm 1\%$ así como un ruido de salida de $0.003\% V_{out}$ entre 10Hz y 10kHz. En la figura 3.10 se muestra el esquema eléctrico utilizado, que es una variación del circuito recomendado por el fabricante en su *datasheet* [14].

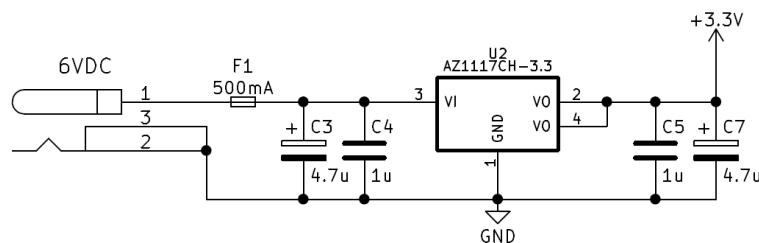


Figura 3.10: Esquema de alimentación a 3.3V

Alimentación 5V

Los 5V serán utilizados por el dispositivo de almacenamiento USB para su alimentación y por el ADS para generar los distintos voltajes de referencia que necesita para operar correctamente. Al afectar de forma directa a las mediciones realizadas por el ADS la eliminación de variaciones en este voltaje es crucial, pues estas supondrán un ruido añadido a la señal final.

El regulador encargado de proporcionar **5V** es el **MCP1711**. Este integrado está caracterizado por tener un rizado de salida menor al 1% del voltaje de salida así como de poseer una corriente en reposo muy baja. Este último parámetro facilitará el diseño de un sistema portátil alimentado por baterías aumentando la duración de la misma. La figura 3.11 muestra el esquema eléctrico utilizado.

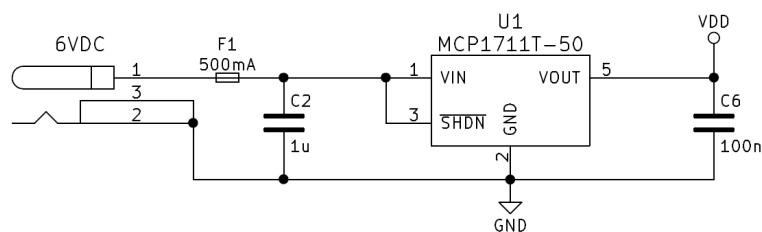


Figura 3.11: Esquema de alimentación a 5V

Para funcionar correctamente ambos integrados necesitan que el voltaje de entrada sea superior al voltaje de salida en un factor que en la documentación técnica recibe el nombre de “Tensión de Dropout”(V_{DROP}).

De acuerdo al *datasheet* de ambos reguladores, V_{DROP} es para 3.3V y 5V, 1.3V y 0.43V respectivamente. Con esa información se deduce que el integrado que limita el diseño es el regulador de 5V de modo que con una entrada de al menos 5.43V ambos reguladores deberían funcionar correctamente.

Finalmente se ha optado por una **fuente de alimentación de 6V** ya que esto permitirá hacer uso de pilas o baterías para alimentar el sistema, dotándolo de independencia de la red eléctrica y eliminando el riesgo de electrocución.

3.3. Circuito electrónico y esquemáticos

El último paso de la fase de desarrollo será generar un **esquemático** capaz de englobar todos los elementos anteriormente presentados, interconectarlos y dar como resultado un sistema funcional.

En este punto es importante valorar dos alternativas de diseño, cada una con sus ventajas e inconvenientes: implementación de todo el circuito de cero o crear una placa que se conecte a la ya existente.

Si bien es cierto que para un diseño final crear una placa que englobe todos los componentes sería lo ideal, pues presentaría un formato más compacto y mejor presentación, hacerlo también supone crear una placa más grande y desaprovechar aquellas ya construidas en proyectos anteriores.

Teniendo en cuenta que se cuenta con varias tarjetas ya montadas y que el sistema está en fase de prototipo, se va a optar por la segunda opción, creando una **segunda tarjeta independiente** en la que se incluirán todos los elementos correspondientes a la gestión de las señales digitales del sistema, dejando las señales analógicas en la otra tarjeta. De esta forma la fase de diseño de la PCB y de montaje se simplifica considerablemente y se **abaratán costes al reutilizarse componentes**.

Para la conexión con la otra tarjeta se aprovechará el espacio dejado por el módulo ESP12-E, pues para comunicarse con el ADS sólo es necesario el bus SPI y todas las señales necesarias se encuentran accesibles desde los conectores de dicho módulo.

El sistema embebido en esta placa incluirá los elementos que se pueden ver en la figura 3.12

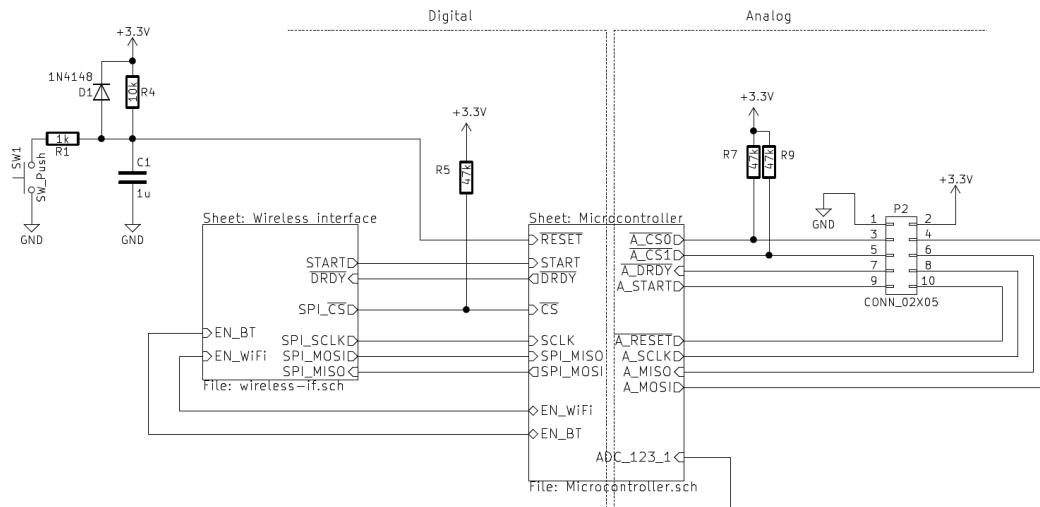


Figura 3.12: Esquema general del sistema

Se ha incluido en la placa un **botón de reinicio** con su correspondiente circuito electrónico así como una realimentación desde la alimentación hasta uno de los pines ADC el microcontrolador para poder **monitorizar el estado de la batería**.

3.3.1. Circuito de alimentación

Aunando los dos circuitos presentados en la sección 3.2.2 el resultado final es el obtenido en la figura 3.13.

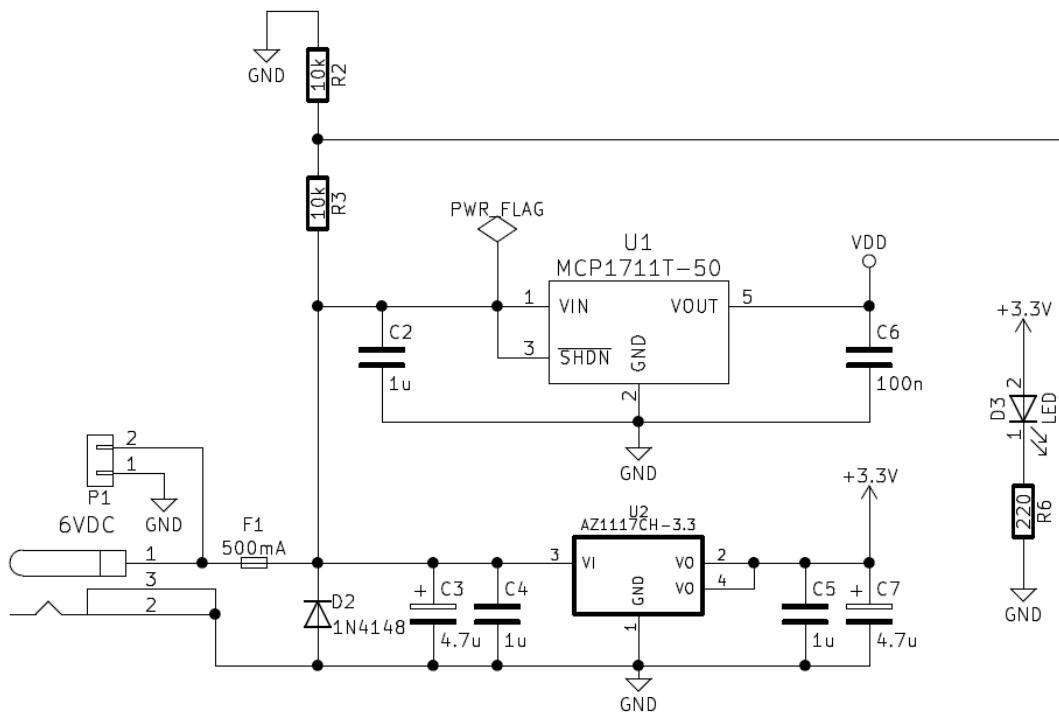


Figura 3.13: Esquemático final del circuito de alimentación

El **fusible** (F1) garantiza que la máxima corriente que consumirá el dispositivo es de 500mA, **evitando** así que el sistema o el paciente sufran **daños** en caso de un cortocircuito. Adicionalmente se ha incluido un diodo LED cuya función es indicar el estado de la placa. Si la placa ha encendido correctamente o si se encuentra en funcionamiento el LED se encenderá.

El diodo (D2) junto con el condensador (C3) evitarán que los transitorios afecten al voltaje de entrada asegurando así que la alimentación que recibirán ambos integrados será lo más estable posible.

Con el objetivo de **monitorizar el estado de la batería** se ha utilizado el **ADC incorporado en el propio microcontrolador**. Como la señal de entrada se encuentra fuera del rango soportado en las especificaciones del STM se ha optado por incorporar las resistencias R2 y R3 formando un divisor de tensión utilizando así la señal resultante para realizar las medidas.

3.3.2. Microcontrolador

El **microcontrolador es el núcleo del sistema**. Este hace de **centro de control** de todas las señales digitales que se transmiten así como de **gestor de dispositivos**, decidiendo que dispositivos se encuentran activos en cada momento. Para gestionar que dispositivos se encuentran habilitados se ha sustituido el *jumper* de la placa original por GPIO del microcontrolador. De esta forma se consigue mayor flexibilidad, automatización y se optimiza el consumo.

Adicionalmente se ha integrado un LED conectado al pin PA9 que dotará a la placa de indicadores visuales del estado en el que se encuentra.

La figura 3.14 muestra una representación del integrado junto con todos los elementos con los que está conectado.

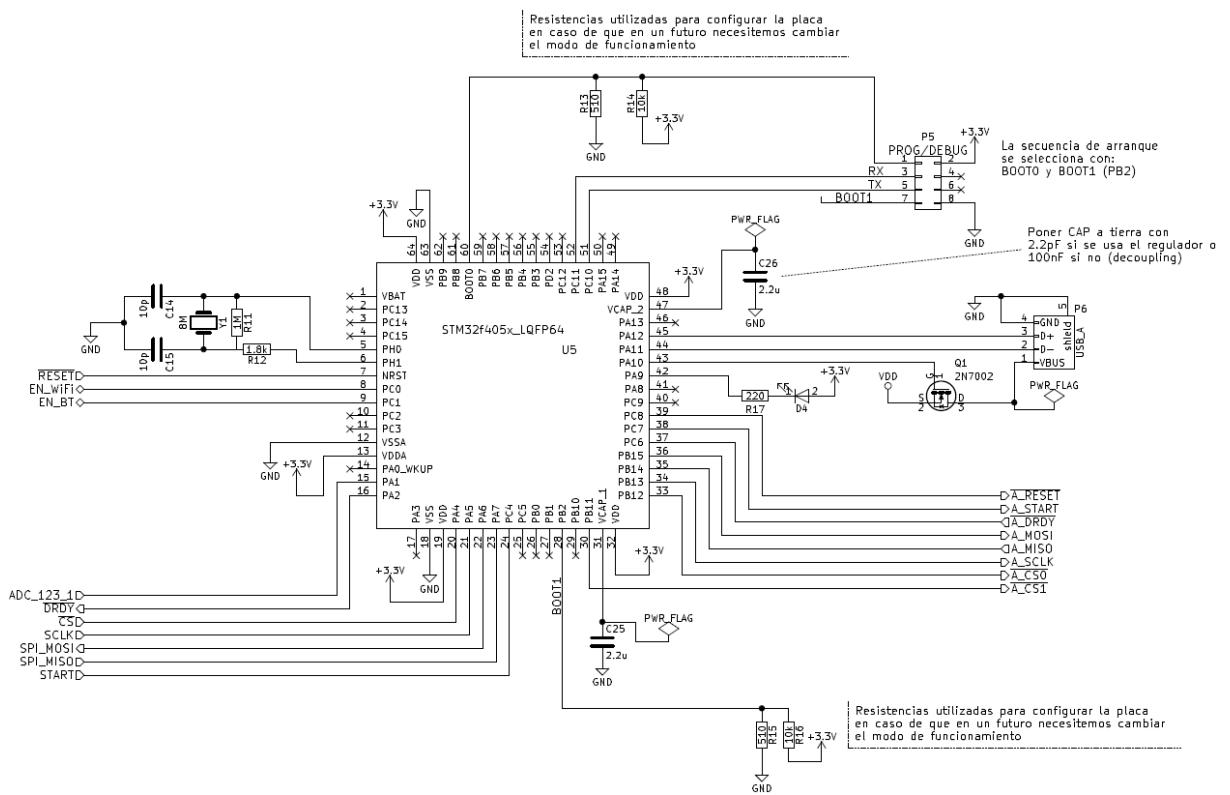


Figura 3.14: Esquemático del microcontrolador

El **modo de arranque** del dispositivo viene determinado por el estado de los pines Boot0 y Boot1. Haciendo uso de las resistencias R13, R14, R15 y R16 se consigue forzar que en condiciones normales de operación el STM **arranque desde la memoria Flash integrada**.

Con el objetivo de **reprogramar el microcontrolador** se ha añadido el conector P5. Dicho conector permite alternar entre los distintos modos de arranque del STM e interactuar con él por UART. Esta última característica será la que permitirá subir el código a ejecutar pero también brinda funciones de *debug*.

El sistema aprovecha dos de los tres buses SPI. El bus SPI1, capaz de transmitir a 41Mb/s, se ha reservado para la comunicación ESP12-E \rightleftarrows STM32F4 mientras que el utilizado para comunicarse con los ADS es el SPI2, con una velocidad de hasta 21Mb/s.

Aunque el MCU puede trabajar con un oscilador interno, se ha optado por la utilización de un **oscilador externo basado en un cristal de cuarzo**. Esa opción permite una mayor precisión en el reloj y la posibilidad de usar PLL para aumentar la frecuencia de trabajo del procesador.

El circuito asociado al cristal se puede observar en la parte superior izquierda de la figura 3.14 pero se incluye a continuación para facilitar la lectura de este documento:

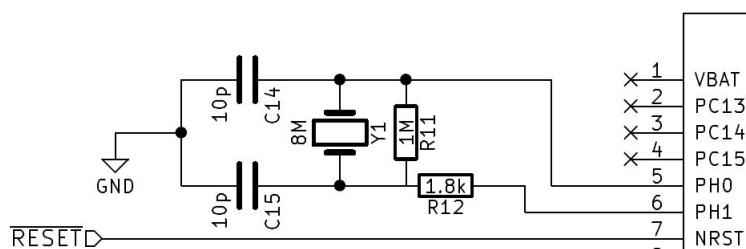


Figura 3.15: Detalle del circuito asociado al oscilador externo

Para realizar un diseño óptimo del circuito del oscilador se ha utilizado como orientación la hoja de características del dispositivo junto con una guía de buenas prácticas [15], ambos proporcionados por el fabricante. La configuración utilizada a nivel de software para la gestión de dicho reloj se explicará en mayor profundidad en el capítulo 5.

Se han añadido todos los condensadores recomendados por el fabricante. Algunos deben tener una capacidad determinada en función del modo de funcionamiento del microcontrolador (C25 y C26), otros, denominados **condensadores de desacoplo**, tienen como objetivo eliminar el ruido de altas frecuencias de la zona de alimentación. Un condensador a resaltar es el C11, denominado **Bulk** y su función es garantizar una alimentación lo más estable posible.

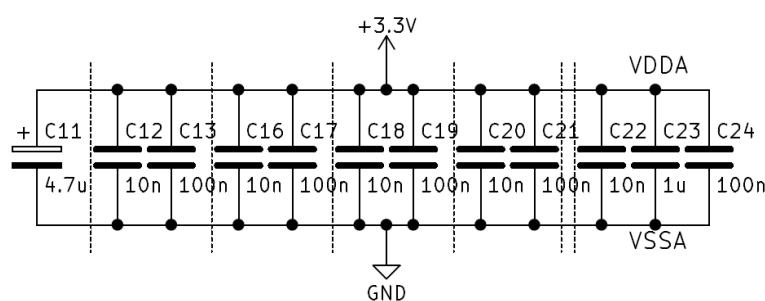


Figura 3.16: Condensadores Bulk (C11) y de desacoplo

Por simplicidad y legibilidad se han agrupado todos los condensadores en una zona del esquemático, pero a la hora de diseñar la PCB será necesario tener presente que para que el dispositivo funcione correctamente estos últimos **deben localizarse lo más cerca posible de los pines de alimentación**.

Finalmente, como el MCU tiene la capacidad de interactuar directamente con dispositivos USB, con el objetivo de implementar en un futuro características que aprovechen dicha capacidad se ha añadido un conector USB cuyo bus de alimentación se encuentra controlado por el propio MCU. De esta forma es posible habilitar y deshabilitar el dispositivo USB y reducir el consumo.

3.3.3. Interfaz inalámbrica

Para finalizar con la parte del diseño, se incluirá una explicación de los esquemáticos necesarios para hacer funcionar los dos microcontroladores encargados de transmitir la información a través de Wifi y Bluetooth.

ESP12-E

El ESP necesita, al igual que el STM, condensadores Bulk y de desacoplo. Ambos se pueden apreciar en la figura 3.17 en la parte inferior izquierda y deberán estar localizados en la PCB lo más cerca posible de los pines de alimentación.

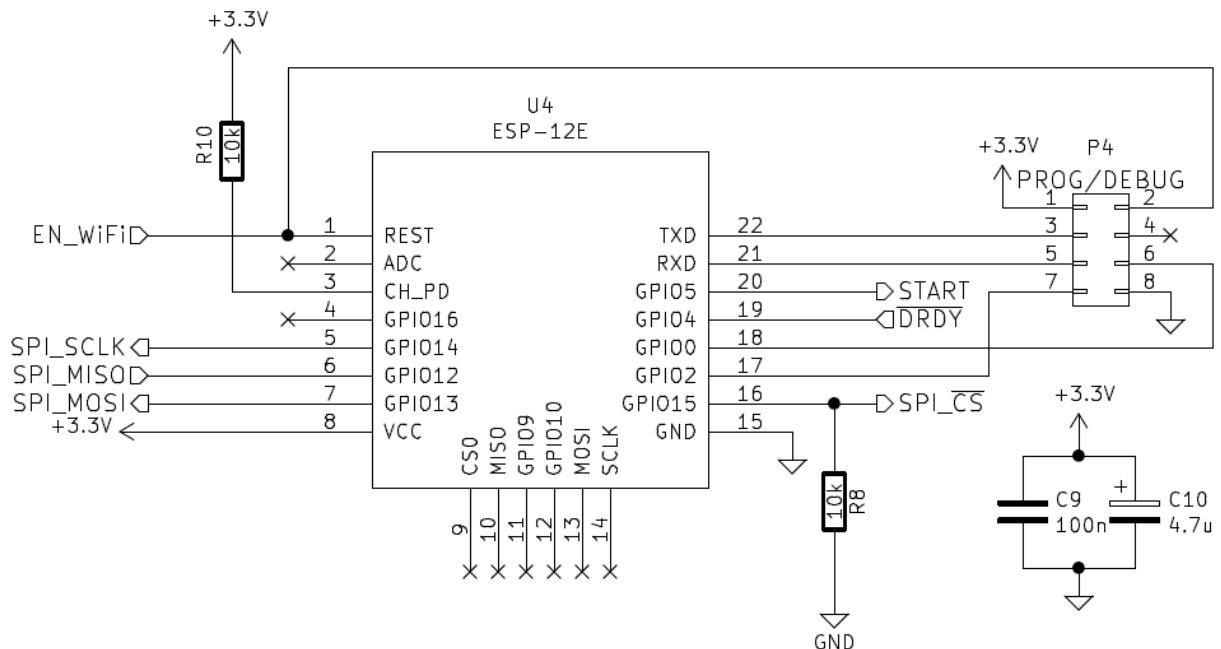


Figura 3.17: Esquemático del ESP-12E

La resistencia R10 fuerza al ESP a estar en un estado activo salvo que el STM (Máster del sistema) lo des habilite haciendo uso del pin “EN_WIFI”.

El modo de arranque del ESP viene definido por la resistencia R8 junto con el conector P4. Para poder ponerlo en modo programación sólo será necesario cortocircuitar los pines 6 y 8. El acceso al bus UART se realizará a través de los pines 3 y 5.

Bluetooth Simblee

Para el dispositivo Bluetooth se ha incluido otro conector de programación. En esta ocasión sólo serán necesarios los pines 2, 3 y 4 para la programación ya que el IDE de Arduino se encargará de la gestión de los diferentes modos de arranque el dispositivo.

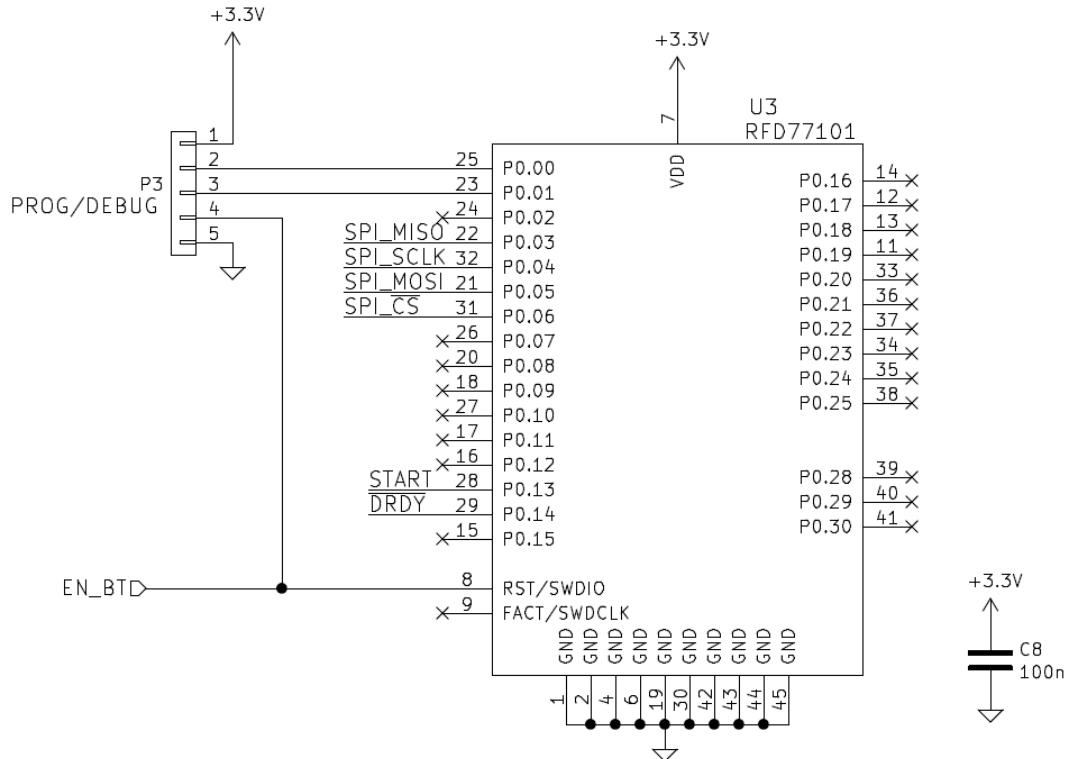


Figura 3.18: Esquemático del dispositivo Bluetooth Simblee

El bus SPI así como los de DRDY y START han sido conectados al STM ya que dichas conexiones serán necesarias para la transmisión de datos entre el STM y el módulo Bluetooth.

En esta ocasión se ha dejado sólo un condensador de desacoplo dado que los de *Bulk* dispuestos para los otros dispositivos suplen la necesidad de utilizar otro para este.

Todos los esquemáticos presentados a lo largo de este capítulo han sido generados haciendo uso de la herramienta KiCad. Al ser un esquemático, su lectura e interpretación es independiente de la herramienta siendo el elemento “PWR_FLAG” (presente en algunas figuras) el único exclusivo de dicha herramienta. Este elemento sirve para evitar errores, ya que todos los pines categorizados como pines de alimentación que no lleven un “PWR_FLAG” provocarán una alerta al compilar y generar el esquemático.

4

Implementación de la PCB

En el diseño e implementación de la PCB es muy importante tener presente que cualquier fallo a nivel de hardware supone una gran pérdida de tiempo y, por lo tanto, dinero. Desde que el diseño es terminado y se manda a producir la placa hasta que esta se recibe transcurre una media de dos semanas . Perder esa cantidad de tiempo por un fallo de diseño a nivel docente supone, en el peor de los casos, no entregar el proyecto en la fecha acordada, pero a nivel empresarial puede significar perder la exclusividad del diseño cuando se compite con otras compañías.

En el diseño es muy importante revisar cualquier tipo de error con el fin de evitar retrasos en la entrega de proyectos. Una revisión manual puede evitar parte de ellos pero la utilización de una buena herramienta bien configurada contribuye a la realización de esta tarea de una manera más rápida y eficaz. Como ya se ha mencionado en el capítulo 3, para la realización de este proyecto se ha utilizado la herramienta KiCad, no sólo por ser Software Libre sino porque presenta ciertas ventajas o características muy interesantes:

- **Entorno de desarrollo integrado:**

Desde la propia herramienta se pueden hacer los esquemáticos, definir los componentes y librerías e, incluso, diseñar y previsualizar la PCB.

- **Multiplataforma:**

Disponible en Windows, Linux y Mac OS.

- **Respaldado por una gran comunidad**

KiCad tiene una gran comunidad que deja a disposición de los usuarios una documentación muy extensa.

- **Constante desarrollo**

Se liberan actualizaciones con regularidad.

Al abrir la herramienta se carga un gestor de proyectos desde el que se puede gestionar un gran número de elementos.

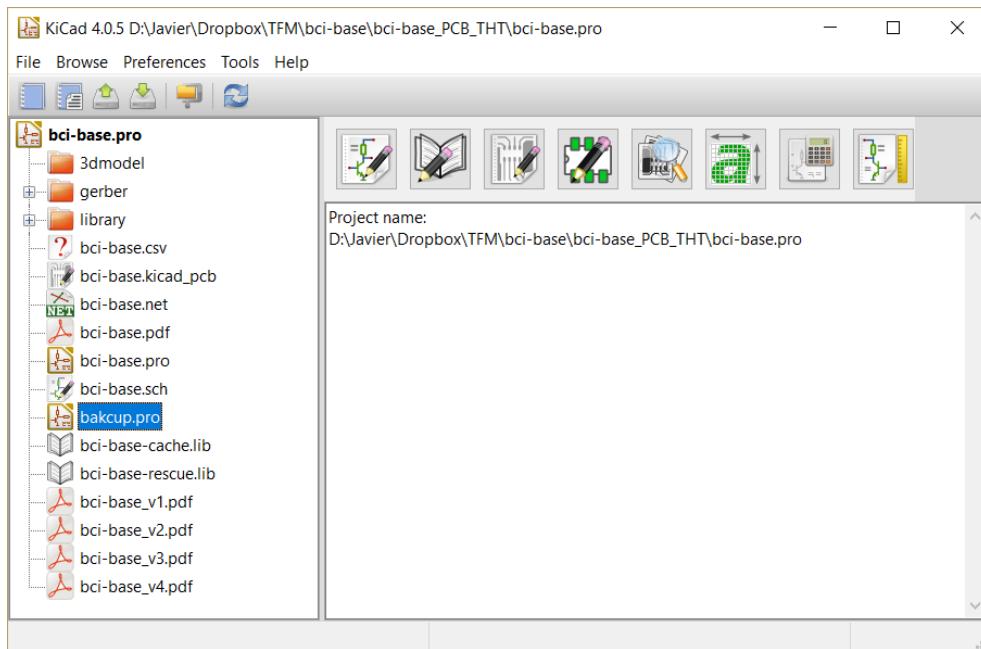


Figura 4.1: Gestor de proyectos de KiCad

Como se puede observar en la imagen, a la izquierda aparecen todos los archivos pertenecientes al proyecto mientras que en la parte superior hay un acceso directo a las distintas partes que componen la herramienta (gestor de esquemáticos, librerías, modelos 3D, etc.).

Durante la fase de diseño y realización del esquemático, cada componente fue seleccionado de entre los disponibles preinstalados en KiCad. Aunque los componentes más comunes como son las resistencias o los condensadores se pueden encontrar sin problemas, para trabajar con otros componentes como el ESP ha sido necesario crear una librería propia. Más adelante, cuando las especificaciones de diseño se hayan definido, se enlazará el símbolo que representa cada componente con el elemento físico que estará presente en la PCB, es decir, su *footprint* y su modelo 3D.

4.1. Limitaciones del fabricante

El diseño de la PCB tiene ciertas **restricciones**, algunas impuestas por el propio diseño del circuito (tipo de componentes, número de pistas, tamaño final, etc.), otras, como las tratadas en esta sección, serán **impuestas por el propio fabricante de PCBs**.

La mayoría de empresas fabricantes de PCBs del mercado dejan a disposición de sus clientes un listado de las limitaciones con las que cuentan, garantizando que cualquier diseño que se adecue a ellas será impreso correctamente.

De entre las disponibles en el mercado se seleccionó la compañía ITEAD por presentar una buena relación prestaciones/precio, un tiempo de impresión y envío bajo y haberse contratado sus servicios con anterioridad.

En su página web [16], ITEAD ha preparado una tabla con un resumen de las características con las que se puede contar si se imprime una PCB en condiciones normales. Dicha tabla se recoge a continuación:

Característica	Valor
Layers	1 - 4
Material	FR-4
Board Dimension (max)	380mm X380mm
Board Dimension (min)	10mm X10mm
Outline Dimension Accuracy	±0.2mm
Board Thickness	0.40mm–2.0mm
Board Thickness Tolerance	±10 %
Dielectric Separation thickness	0.075mm–5.00mm
Conductor Width (min)	0.15mm (Recommend>8mil)
Conductor Space (min)	0.15mm (Recommend>8mil)
Outer Conductor thickness	35um
Inner Conductor thickness	17um–100um
Copper to Edge	>0.3mm
Plated Component	
Plated via Diameter(Mechanical)	0.3mm–6.30mm
Plated Hole Diameter Tolerance(Mechanical)	0.08mm
Unplated Hole Diameter Tolerance	0.05mm
Hole Space(min)	0.25mm
Hole to Edge	0.4mm
Annular Ring(min)	0.15mm
Solder Resist Type	Photosensitive ink
Solder Resist Color	Black, Green, White, Blue, Yellow
Solder Resist Clearance	0.1mm
Solder Resist Coverage	0.1mm
Plug Hole Diameter	0.3mm–0.65mm
Silkscreen line width (mim)	6mil

Tabla 4.1: Restricciones de ITEAD para la fabricación de una PCB

Aunque la lista de restricciones parece alta, la mayoría de ellas no supone un problema para un proyecto de estas dimensiones. Para comprobar si esta afirmación es correcta bastará con comprobar si la característica más restrictiva se cumple, es decir, comparar la separación mínima de los pads del módulo Bluetooth con la separación mínima entre conductores, denominada en la tabla 4.1 como *Conductor Space*. De acuerdo a su *datasheet*, la separación entre pines de este componente es de 9.84 mil (0.25 mm), valor muy superior a los 8 mil (0.2 mm) recomendados por el fabricante.

Adicionalmente, informan de que la utilización de *Buried vias* y *Blind vias* no es posible por el momento.

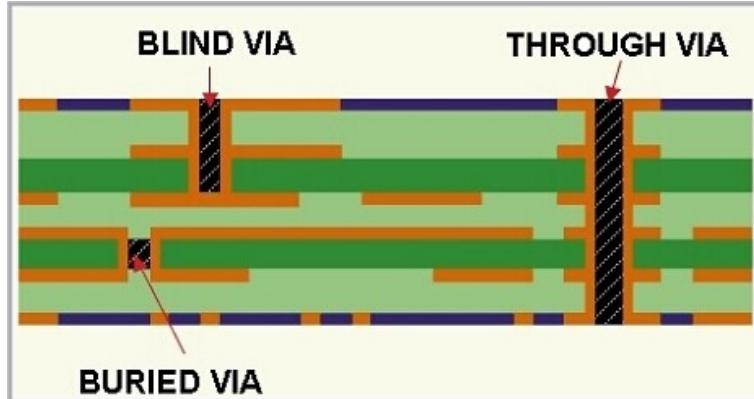


Figura 4.2: Tipos de vías

Con toda la información recopilada hasta el momento ya es posible configurar KiCad para forzar que dichas restricciones se cumplan en todo momento. El menú “*Design Rules*” permite configurar estos parámetros (ver fig. 4.3b) así como realizar ciertos ajustes dependiendo de la función que tendrá cada pista (ver fig. 4.3a).

(a) Condiciones especiales

	Clearance	Track Width	Via Dia	Via Drill	uVia Dia	uVia Drill
Default	0.008	0.008	0.028	0.012	0.028	0.012
POWER	0.008	0.008	0.028	0.012	0.028	0.012
Vdd	0.008	0.011811023E	0.028	0.012	0.028	0.012

Membership:

Net	Class
/Microcontroller/ADC_123_1	Defau
/Microcontroller/A_MISO	Defau
/Microcontroller/A_MOSI	Defau
/Microcontroller/A_SCLK	Defau
/Microcontroller/A_START	Defau
/Microcontroller/BOOT1	Defau
/Microcontroller/EN_BT	Defau
/Microcontroller/EN_WiFi	Defau
/Microcontroller/SLCK	Defau
/Microcontroller/SPI_MISO	Defau

(b) Reglas de diseño en KiCad

Via Options:

- Do not allow blind/buried vias
- Allow blind/buried vias

Micro Vias:

- Do not allow micro vias
- Allow micro vias

Minimum Allowed Values:

- Min track width ("": 0.008)
- Min via diameter ("": 0.028)
- Min via drill dia ("": 0.012)
- Min uvia diameter ("": 0)
- Min uvia drill dia ("": 0)

Custom Via Sizes:

Drill value: a blank or 0 => default Netclass value

Via 1	Diameter	Drill
Via 1		
Via 2		
Via 3		
Via 4		
Via 5		POWE
Via 6		Vdd
Via 7		Vdd
Via 8		

Custom Track Widths:

Track 1	Width
Track 1	
Track 2	
Track 3	
Track 4	
Track 5	
Track 6	
Track 7	
Track 8	

Figura 4.3: Reglas de diseño en KiCad cumpliendo las restricciones de ITEAD

4.2. Componentes y librerías

Aunque en el esquemático se escogieron los valores de todos los componentes, a la hora de realizar un diseño final hay que tener en cuenta otros muchos parámetros. ¿Qué tamaño tendrá el componente?, ¿cuál será su tolerancia?, ¿qué formato se ajusta mejor, THT o SMT? La respuesta a todas estas preguntas acabará definiendo el componente a elegir, su precio y su disponibilidad.

Utilizar componentes en el formato THT puede suponer una ventaja las primeras veces que se realiza una soldadura o al trabajar con electrónica de potencia. En ambos casos se aprovecha el grosor del conector y el hecho de que atraviese la placa para dar mayor comodidad al técnico y disminuir la resistividad de la unión respectivamente.

Por desgracia, al trabajar con electrónica digital o analógica que no involucra alta potencia, la utilización de dichos componentes limita el diseño e impone restricciones que mediante SMT se evitan con relativa facilidad. Un claro ejemplo es la imposibilidad de enrutar pistas bajo los conectores de dichos componentes THT.

A lo largo del diseño de la PCB se seleccionará en la medida de lo posible componentes en su formato **SMT** ya que además introducen menos ruido en el circuito.

Por simplicidad y comodidad se ha escogido trabajar con una tamaño estándar de **0603** con medidas de **0.063" x 0.031"** (**1,6 mm x 0,8 mm**) ya que, por un lado permiten su manejo y soldado sin necesidad de herramientas especiales y, por otro, son medidas muy comunes facilitando la localización de componentes así como de distribuidores primarios y secundarios.



Figura 4.4: Resistencia en formato 0603 [17]

4.2.1. Asignación de *footprint*

Tras decidir el tamaño de los componentes se debe **enlazar** cada uno de los **símbolos** presentes en el **esquemático** que se generó al comienzo del proyecto su ***footprint*** correspondiente. Adicionalmente pueden añadirse **modelos 3D** para una visualización más realista del acabado final.

Si bien el segundo elemento no es normalmente necesario, el primero resulta imprescindible pues contiene información básica sobre el componente (número de pads y conexiones, tamaño real, serigrafías, etc).

Como ya se ha mencionado anteriormente, KiCad se caracteriza por tener una gran comunidad que lo respalda. Esto se traduce en que la mayoría de los componentes del mercado ya han sido incluidos en **librerías de código abierto** disponibles en Internet, lo cual supone un ahorro considerable de tiempo de cara a este o futuros proyectos. Por desgracia esto no es siempre suficiente por lo que KiCad ha sido diseñado para, en caso de ser necesario, crear una librería en la que almacenar los *footprints*, serigrafías y **modelos 3D propios**, conteniendo las **herramientas con las que crear o editar** uno ya existente para adaptarlo a las necesidades del proyecto.

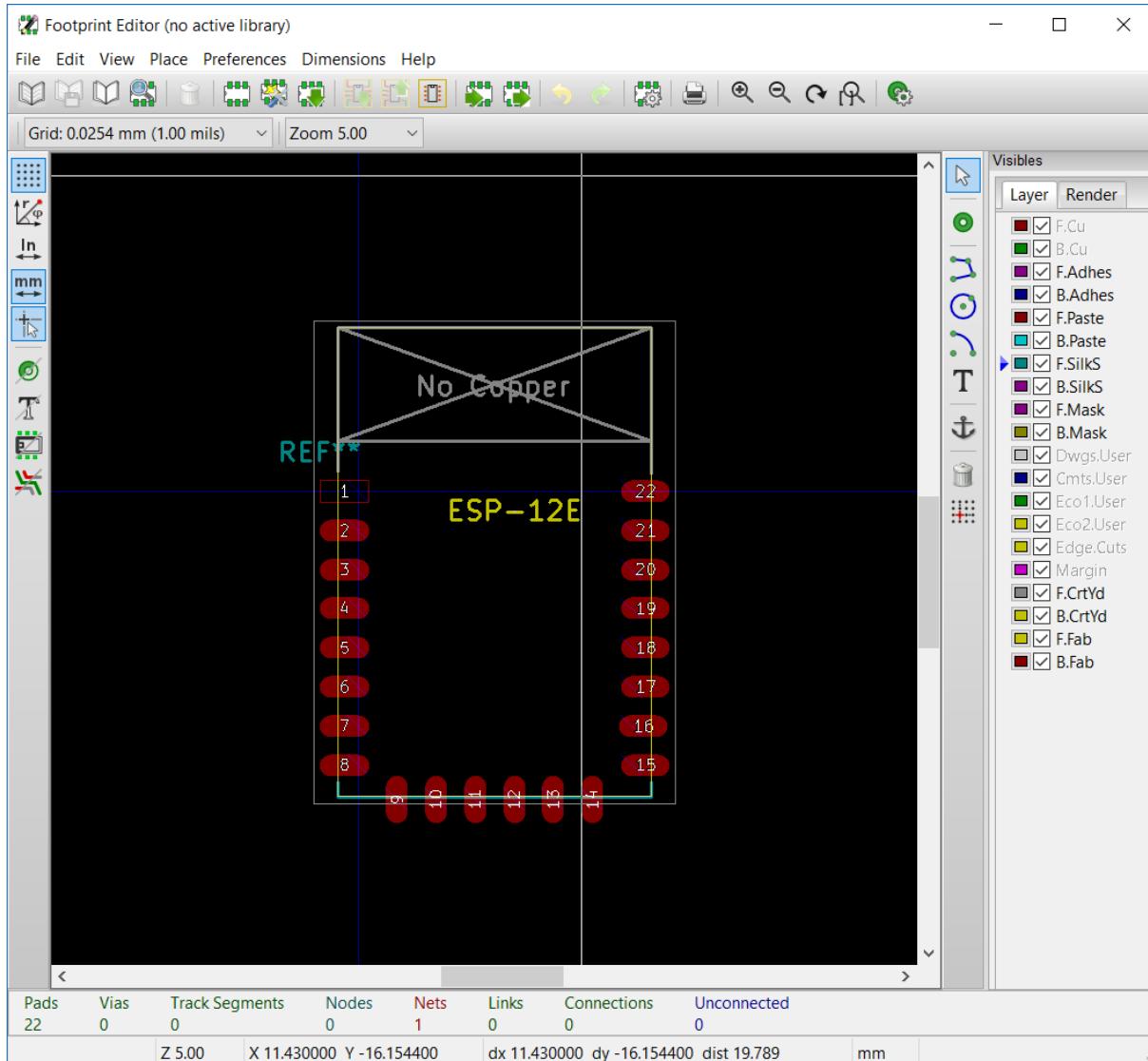


Figura 4.5: *Footprint* modificado para adaptarlo al ESP12-E

Una vez se han localizado o creado los *footprints* de todos los componentes es necesario asociarlos entre si. Esto permite que en la fase de diseño de la PCB, KiCad pueda cargar una representación física del componente. La asociación se realiza con el complemento CvPcb, accesible desde Eeschema. La figura 4.6 muestra la interfaz de usuario tras finalizar este proyecto.

En la imagen 4.6 se puede observar a la izquierda las librerías disponibles, en la columna central los componentes de este proyecto y a la derecha los *footprints* contenidos en la librería seleccionada en la columna izquierda.

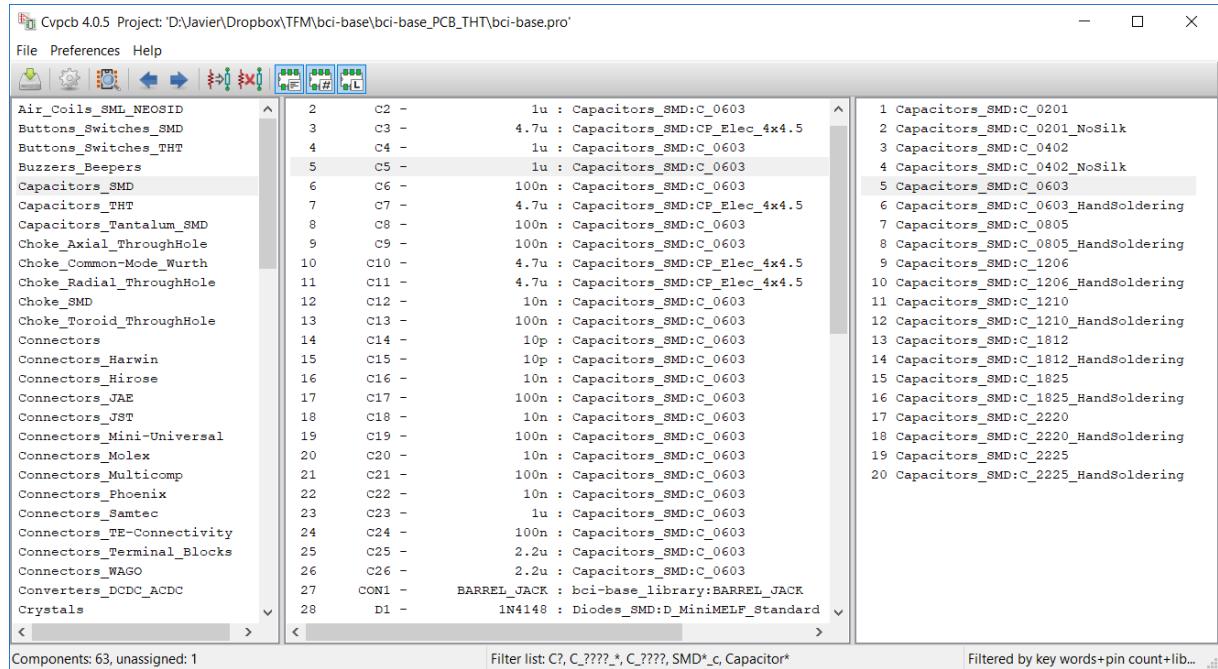


Figura 4.6: Herramienta cvpcb asociando a un componente su *footprint*

Para agilizar el proceso es posible realizar **búsquedas** de componentes, *footprints* y librerías haciendo uso de los tres botones situados en la parte superior derecha. De izquierda a derecha permiten **filtrar** por palabras clave extraídas del componente, por **número de pines** y por **librería activa**.

4.3. PCBnew

Eeschema da como salida dos tipos de archivos que representan la interconexión del los elementos del circuito. Por un lado, un archivo PDF destinado a su lectura por humanos, por otro, un fichero en formato NET. Este último será el fichero de entrada de la herramienta de diseño de PCB denominada **PCBnew**.

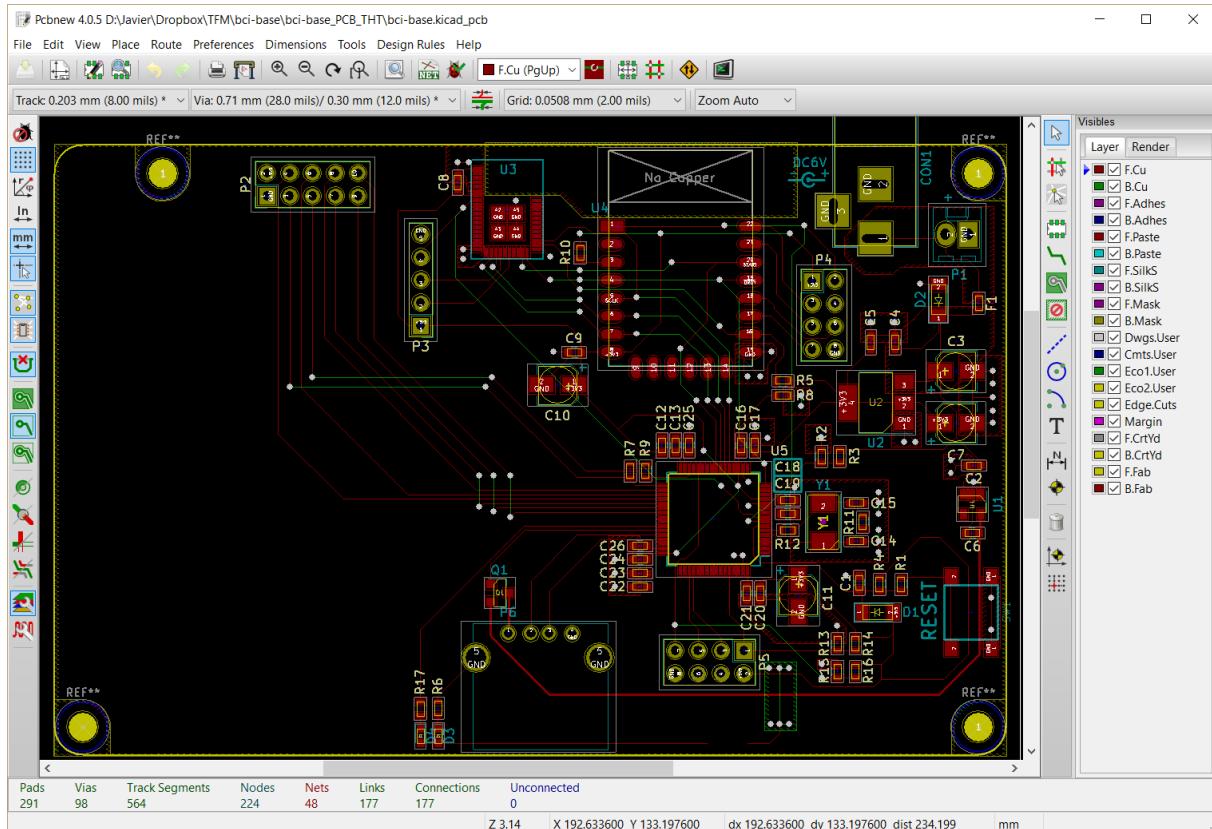


Figura 4.7: Panel principal de la herramienta PCBnew

Tras cargar el .net todos los componentes aparecen representados en la zona de trabajo. Cada uno de los conectores muestra una línea que indica a que otros dispositivos se debe conectar para que el la *netlist* se generada por Eeschema se cumpla.

Para proyectos de menor envergadura como este, la disposición de los **elementos** se puede realizar de **forma manual**, pero KiCad está pensado para poder trabajar con diseños de miles de componentes. En estos casos es posible utilizar la propia **disposición automática** de KiCad o importar la disposición de herramientas externas como **FreeRoute**.

Una vez posicionados todos los componentes se deben realizar las interconexiones definidas en el esquemático. Con este **objetivo** se utiliza el **trazado de las pistas**. Como ya se había mencionado, las características de las pistas vendrán determinadas por las reglas de diseño y el tipo de red al que pertenece (por defecto, de alimentación, de tierra, etc).

Para aquellos puntos comunes a varios dispositivos como son **GND** o **Vcc** es posible definir **planos**. La definición de planos permite, además de ahorrar el trazado de un gran número de pistas, **disminuir la resistividad de una conexión**, consiguiéndose así que el voltaje sea lo más uniforme posible en todo el sistema.

La definición de un plano no imposibilita el trazado de una pista que lo atraviese. Aquellas pistas que atraviesan un plano son protegidas con una **zona de guarda** sin cobre que evita cortocircuitos y cuyo grosor viene definido en las reglas de diseño.

En ocasiones será necesario forzar que una parte de la placa no contenga ningún **elemento conductor**, bien para evitar apantallamientos como en el caso de las antenas, bien para forzar aislamientos entre dos zonas. Para estos casos se puede definir un plano especial denominado “*keep out area*”. Al contrario que los planos normales, **las pistas no pueden atravesarlos**.

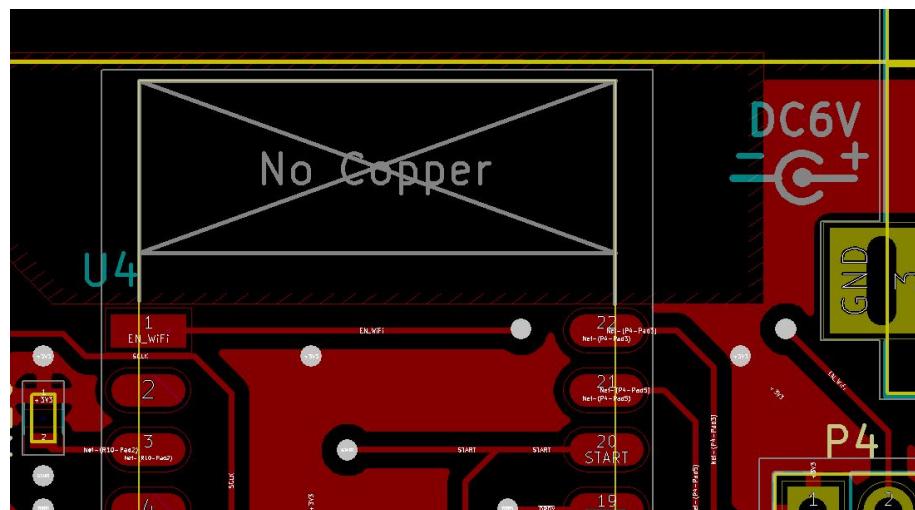


Figura 4.8: Ejemplo de planos, *keep out area* y pistas en la PCB

En ocasiones el propio sistema puede incluir tantos componentes, pistas y planos que unos elementos acaban afectando a la visibilidad. Para evitar estos problemas, en la parte derecha del panel principal de PCBNew hay habilitado un panel que permite activar o desactivar el **renderizado** de ciertos objetos. Es posible **ocultar** cualquier **grupo de elementos**: planos, serigrafías, componentes, etc.

4.4. Diseño final de la PCB

El sistema seguido para la **distribución de componentes en la PCB** ha venido determinado en gran medida por la disposición de componentes de la tarjeta con la que deberá comunicarse. Con el objetivo de minimizar la distancia entre los conectores de ambas placas, la zona encargada de **alimentar el sistema** ha sido situada en la **zona superior derecha de la placa**. La localización del conector que transmitirá las señales a la tarjeta de adquisición también ha sido escogida siguiendo esta premisa. Tras posicionar los elementos más problemáticos, el resto de los componentes se han distribuido de manera que el número de cruces sea mínimo, evitando así en la medida de lo posible la interferencia entre pistas y simplificando la PCB.

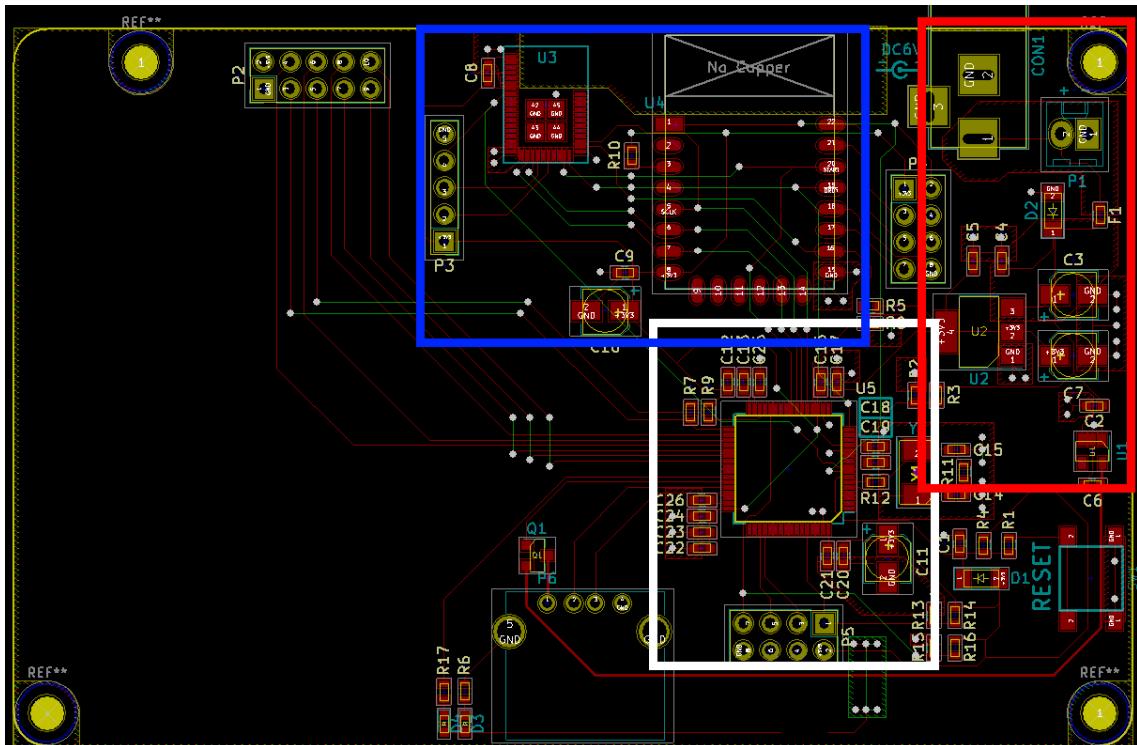


Figura 4.9: Distintas partes que componen la PCB. Alimentación (rojo), microcontrolador (blanco), interfaz inalámbrica (azul).

En las siguientes secciones se explicará con más detalle cada una de las partes de la PCB, haciendo una división en los tres grandes grupos que la componen: **alimentación** (rojo), **microcontrolador** (blanco) e **interfaz inalámbrica** (azul).

4.4.1. Alimentación

La **alimentación** ha sido agrupada para facilitar la **depuración de errores** y la sustitución de componentes así como la simplificación de la conexión con la tarjeta de adquisición. Los conectores se han posicionado en lugares en los que resulte cómodo realizar la interconexión a la par que se mantiene al mínimo la cantidad de cable necesario para ello. Se ha situado el fusible de tal forma que en caso de ser necesaria su sustitución haya disponible suficiente espacio para maniobrar con comodidad.

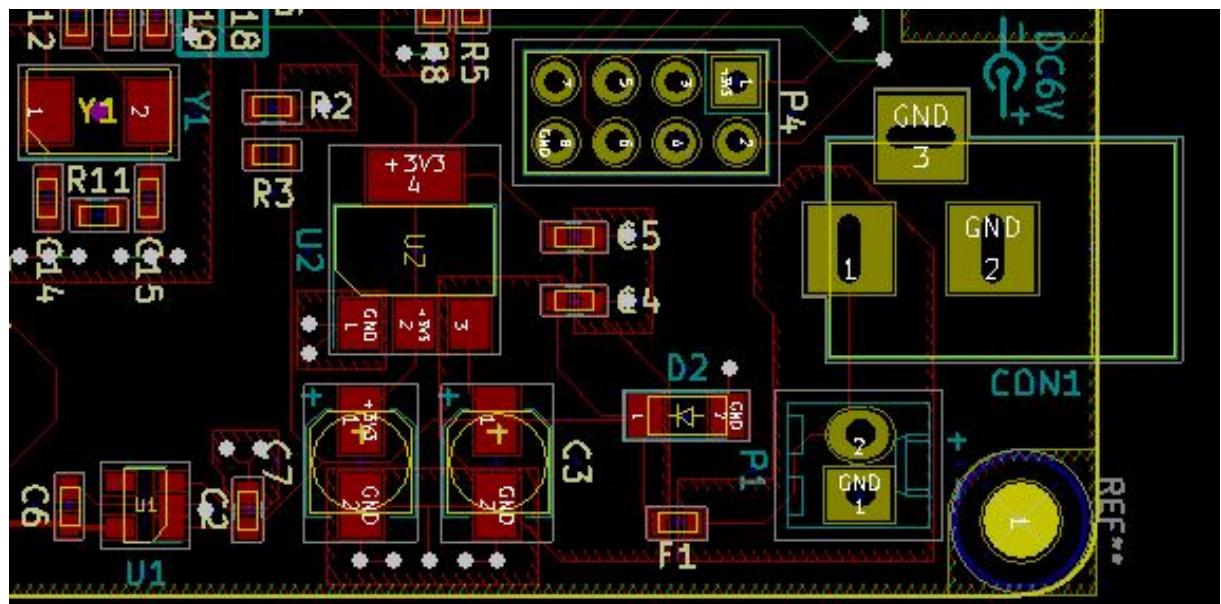


Figura 4.10: Detalle de la alimentación de la placa

4.4.2. Microcontrolador

Durante el posicionado del microcontrolador es muy importante tener en cuenta las restricciones impuestas por el fabricante, pues algunas de estas no se ven reflejadas en la *netlist* ni en el esquemático. Los **condensadores**, tanto de desacoplo como *Bulk*, han sido **colocados lo más cerca posible de los pines de alimentación** para asegurar el correcto funcionamiento del microcontrolador.

STM Electronics ha puesto a disposición de sus clientes un manual para el correcto diseño y puesta a punto del circuito resonador de un microcontrolador. Este documento no sólo recomienda los valores de los componentes a utilizar en dicho diseño, sino que propone pautas a la hora de disponer los componentes en la PBC. Algunas de las recomendaciones son **aislar el circuito del oscilador** del resto de la placa con un plano de tierra, forzar una **distribución** de los componentes lo más **simétrica** posible y, al igual que con los condensadores, **colocar el circuito lo más cerca posible de los pines** del microcontrolador.

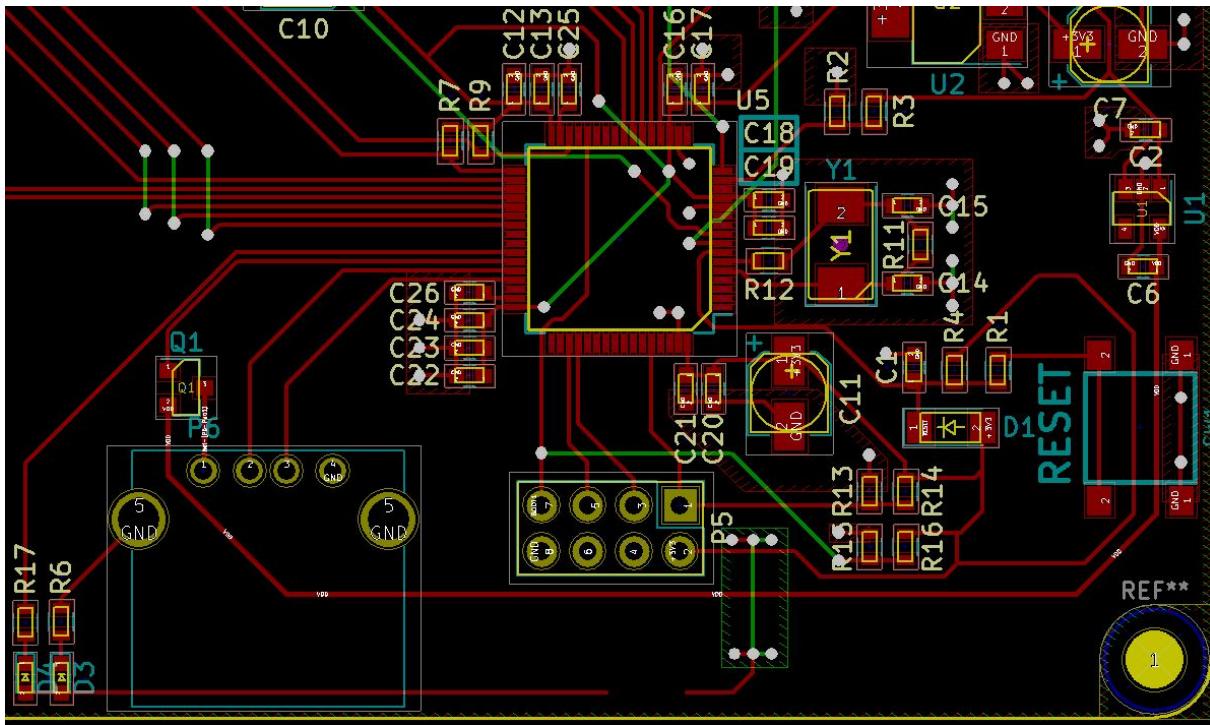


Figura 4.11: Detalle del microcontrolador y los componentes asociados

Se ha dotado a la placa de un circuito que fuerza el **reinicio del microcontrolador**. Dicho circuito será especialmente útil posteriormente ya que, si este se programa correctamente, permitirá reiniciar **todos los microcontroladores simultáneamente**. Adicionalmente, durante el desarrollo del firmware del microcontrolador, se podrá forzar su **arranque en modo reprogramación** sin necesidad de recurrir a cortar la alimentación de todo el sistema.

4.4.3. Interfaz inalámbrica

La zona de la PCB que corresponde a la interfaz inalámbrica ha sido configurada de tal forma que las **antenas** de los distintos módulos se quedan en la **zona exterior** de la placa. Además, para evitar apantallamientos, el plano que se encuentra bajo dichas antenas ha sido eliminado usando una “**keep out area**”. Ambos módulos están formados por microcontroladores de menor potencia que el presente en el STM de modo que los distintos condensadores de desacoplo y *bulk* han sido dispuestos de forma similar.

La localización de los conectores de programación de ambos módulos ha venido determinada por la posición de sus correspondientes pines. El objetivo ha sido aproximarlos al módulo correspondiente sin que por ello se produzcan cruces entre pistas. Al presentar unas restricciones de diseño leves por parte del fabricante, ha sido posible en ocasiones trazar pistas entre los pines del ESP lo cual ha facilitado sensiblemente la tarea de trazado de rutas y disposición de componentes.

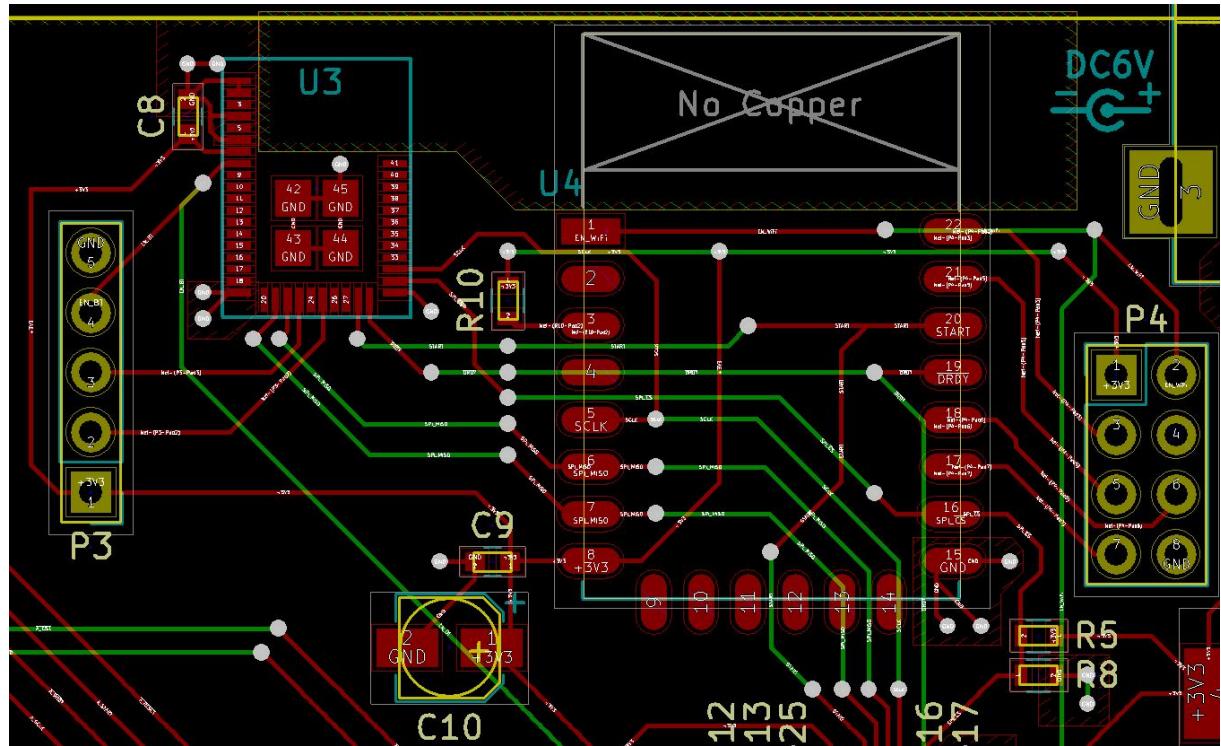


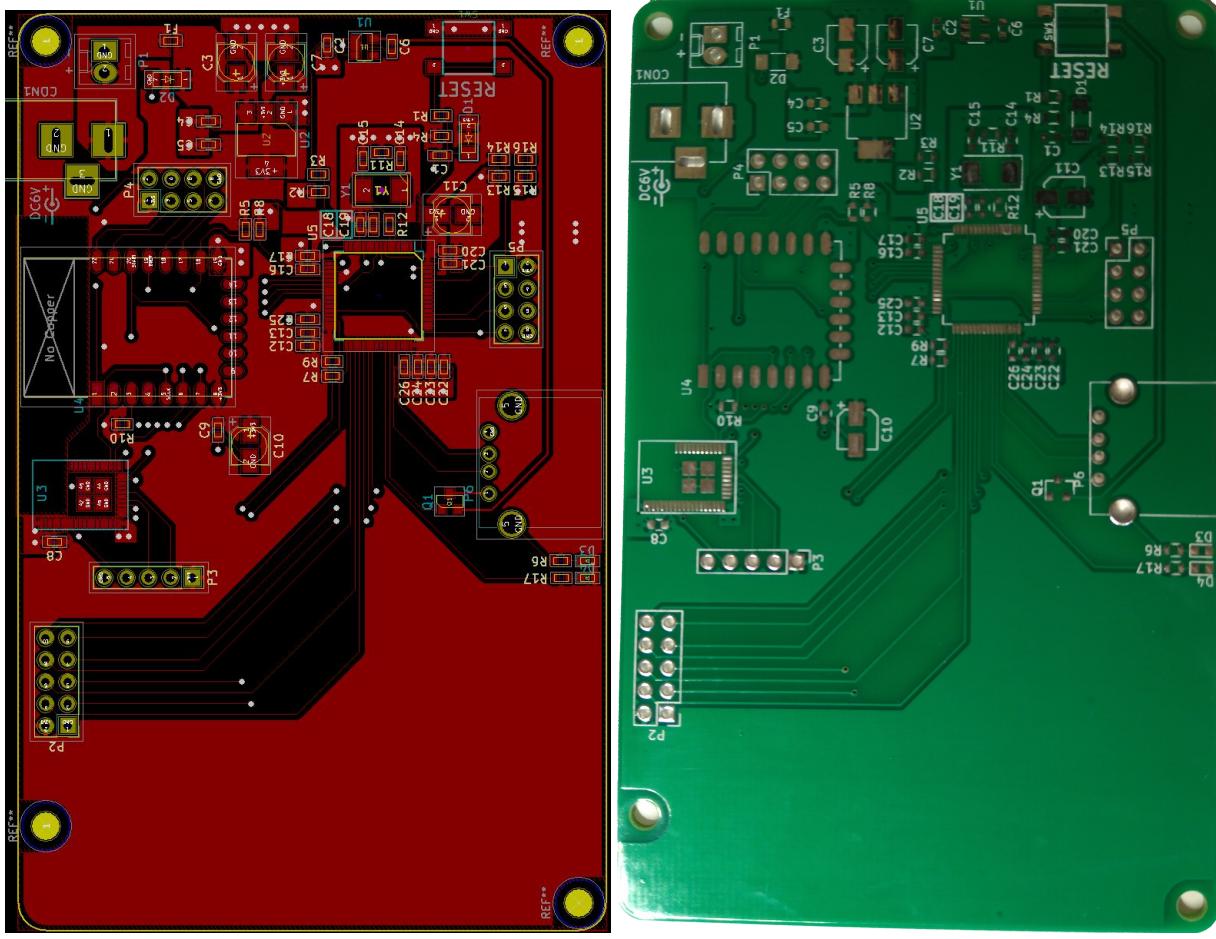
Figura 4.12: Detalle de los módulos WiFi y Bluetooth

Finalmente se han interconectado ambos módulos a través del **bus SPI compartido** así como con el microcontrolador y la alimentación. Con el objetivo de eliminar pistas y de unificar la alimentación se ha definido un **plano de 3.3V** en la parte superior de la placa y un plano de **GND** en la parte inferior.

Por desgracia, debido a la complejidad del diseño y a la cantidad de pistas trazadas, no todas las conexiones entre los elementos se han podido realizar en la capa superior. Por este motivo ha sido necesario realizar una PCB con varias capas y hacer uso de vías, puentes y otras técnicas afines.

4.5. Resultados

A continuación se presenta una comparativa entre el *layout* proyectado utilizando KiCad y el resultado final.



(a) *Layout* en KiCad.

(b) Impresión final

Figura 4.13: *Layout* en KiCad (a) e impresión final (b) de la capa superior

El resultado es bastante positivo. Al haberse respetado todas las restricciones de diseño impuestas por el fabricante, no hay ningún cortocircuito y todas las pistas se han impreso correctamente.

Aquellas zonas en las que no hay conductor en ninguno de los planos aún presenta un soporte físico de fibra de vidrio para dotar de consistencia a la placa.

La **serigrafía** que acompaña a los componentes así como la generada manualmente se aprecia con mayor **claridad** incluso que en la previsualización generada por KiCad.

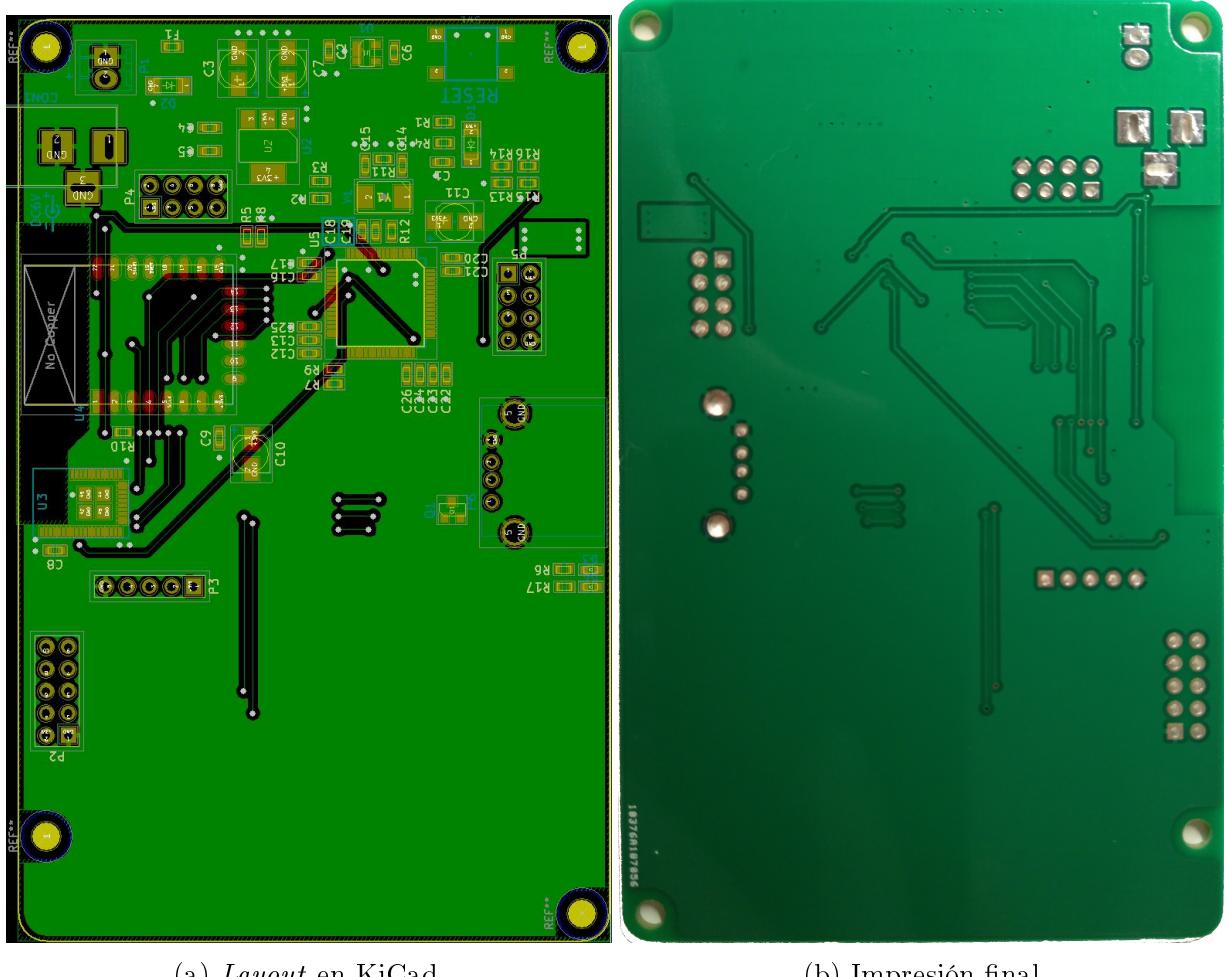


Figura 4.14: *Layout* en KiCad (a) e impresión final (b) de la capa inferior

Como se puede observar, la impresión real se encuentra en **modo espejo** con el *layout* de KiCad de la capa inferior. Este fenómeno ocurre debido a al trabajar con la capa inferior en KiCad se mantiene la **vista cenital** y las capas superiores simplemente no se **renderizan**. Con esto se consigue facilitar el proceso de diseño, no sólo por mantener un punto de referencia constante, sino porque al trazar pistas se puede pasar con comodidad entre unas capas y otras sin tener que cambiar el modo de vista.

4.5.1. Previsualización 3D vs resultado final

Normalmente el *footprint* de un componente incluye una serigrafía que delimita la zona de la placa que ocupará, garantizando que dos componentes cuya serigrafía no se solape no causarán problemas en el montaje final.

Por desgracia, esto no siempre se cumple y en ocasiones puede ocurrir que dos componentes procedentes de una librería poco fiable contengan información errónea.

Para minimizar las posibilidades de error y asegurarse que el resultado final será el deseado KiCad permite realizar una **representación 3D** de todos los componentes así como su posición relativa en la placa.

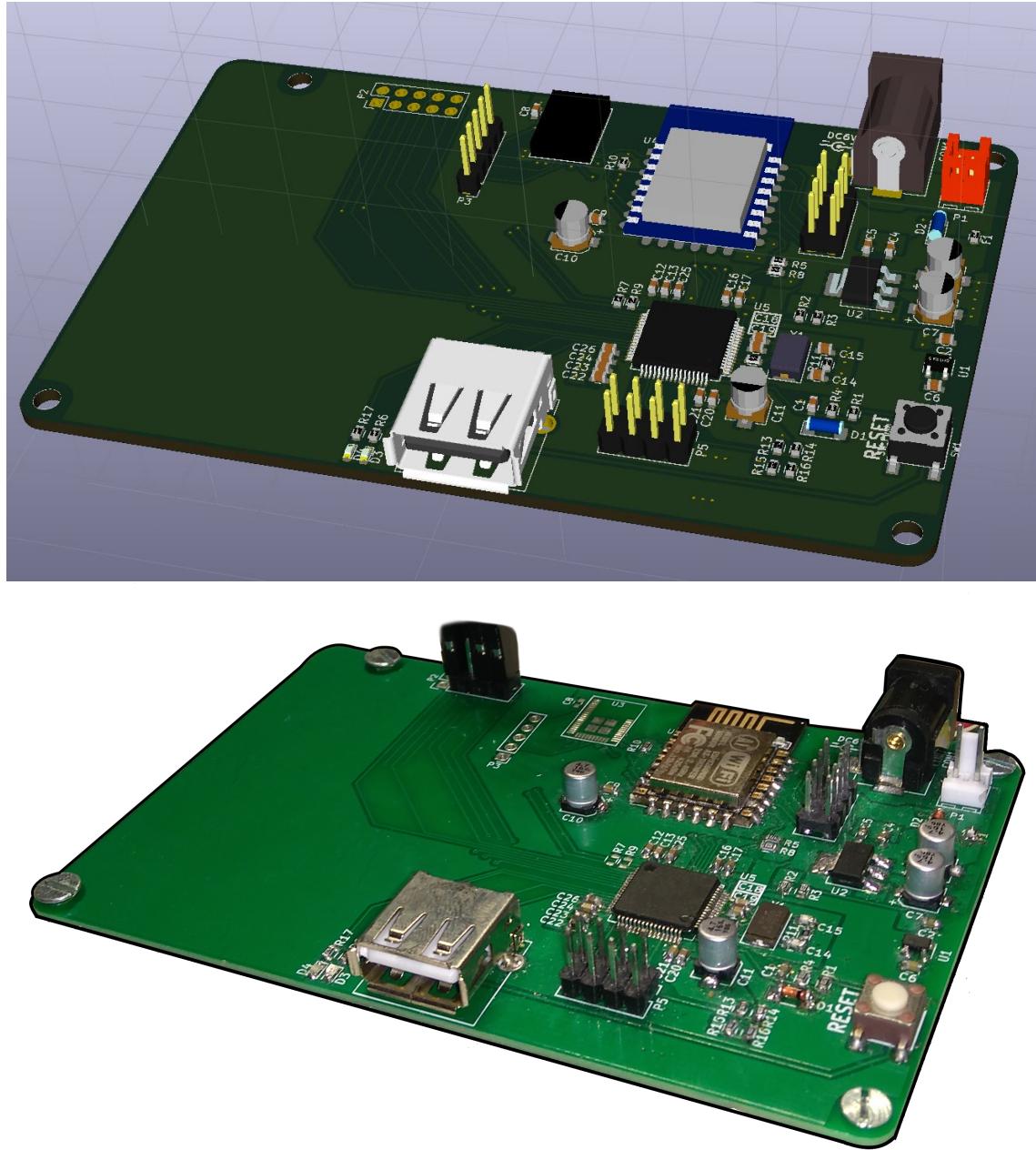


Figura 4.15: Modelo 3D del sistema (arriba) y placa final (abajo)

En la figura 4.15 se puede comprobar que, efectivamente, la **representación 3D proyectada** durante la fase de diseño ha resultado ser **fiel al producto final** obtenido.

5

Implementación del software

Una vez finalizado el soporte físico del sistema será necesario implementar a nivel de software todas las funciones deseadas.

El sistema está compuesto por **tres microcontroladores**, cada uno de ellos con sus propias características y métodos de programación. A lo largo de este capítulo se explicará con detenimiento el software utilizado individualmente para ellos así como los distintos entornos de desarrollo (IDE) necesarios.

Los distintos elementos se presentarán siguiendo el mismo orden que la información que se desea adquirir. Este es:

ADS ⇒ Microcontrolador ⇒ Interfaz inalámbrica ⇒ PC

5.1. Microcontrolador STM32F4

Debido a la complejidad de los microcontroladores de la familia STM32, los desarrolladores han dedicado su tiempo y esfuerzo en crear herramientas que faciliten su programación. Estas herramientas llamadas **IDE** contienen una **recopilación de funciones, utilidades y ejemplos** que simplifican sensiblemente el proceso de diseño e implementación del software que ejecutará el microcontrolador.

Es tan grande la comunidad de desarrolladores que hay detrás del STM que numerosas empresas han visto como una oportunidad de negocio la creación de un IDE propietario.



Figura 5.1: IDEs alternativos disponibles [18]

La figura 5.1 muestra algunos entornos de desarrollo recomendados por el fabricante en su propia página web. Como se puede apreciar se hace distinción entre las opciones comerciales (azul) y las gratuitas (verde).

Uno de los **objetivos** de este proyecto era **utilizar Software Libre** en la medida de lo posible. Por ese motivo se probaron algunas de las alternativas presentes en la imagen anterior dando preferencia a aquellas basadas en Oracle como “System Workbench for STM32” (AC6). A continuación se presentan algunas de las utilizadas al comienzo del proyecto junto con sus características y el motivo de su descarte:

- **CooCox**

Presenta muchos ejemplos prácticos pero su interfaz es muy lenta. Necesita demasiado tiempo para iniciar.

- **Arduino**

Fase muy temprana de desarrollo.

- **System Workbench for STM32**

Buena comunidad pero incompatible con las herramientas de STM.

Finalmente se optó por **armKeil** ya que tiene una gran **cantidad de ejemplos y librerías descargables** desde la propia interfaz, contiene un gran número de **funciones adicionales** y la compatibilidad con las herramientas de STM es muy buena. Aunque en la figura 5.1 aparece entre las alternativas gratuitas se debe destacar que cuenta con dos versiones: una gratuita que permite realizar proyectos básicos y otra comercial, destinada a aplicaciones de mayor envergadura. La **limitación de la versión gratuita** consiste en forzar un tamaño máximo de programa de **32KB**. Si el código a compilar supera dicha longitud directamente no compilará.

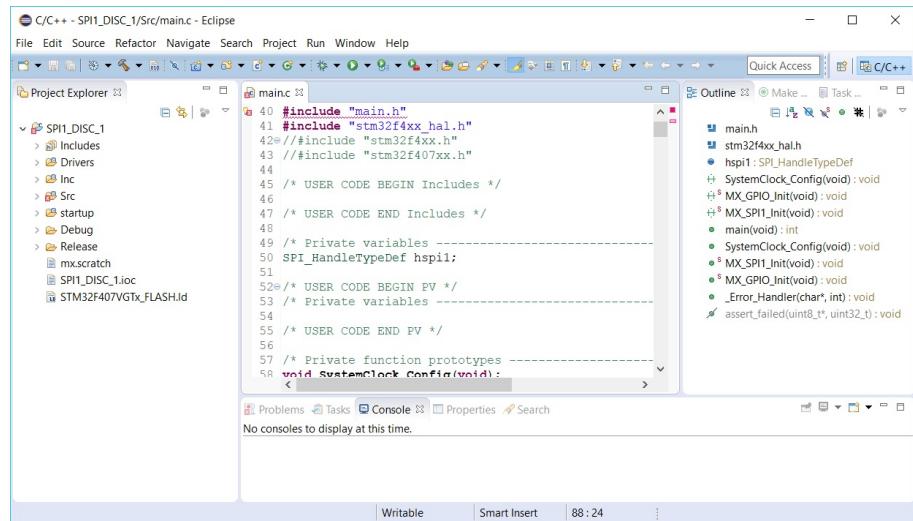


Figura 5.2: Ejemplo de IDE alternativo basado en Oracle (System Workbench for STM32)

5.1.1. Configuración inicial

Aunque Keil permite comenzar un proyecto utilizando como base algunos de los ejemplos que contiene, la configuración de las características del microcontrolador (reloj, interfaces, etc.) puede resultar una tarea muy compleja y tediosa, incluso para aquellos desarrolladores más experimentados. Con el objetivo de facilitar esta tarea, el fabricante ha creado un **software con interfaz amigable** que permite **configurar de forma gráfica el procesador** llamado **STM32CubeMX** que hace uso de un conjunto de elementos y **drivers** llamado **HAL** que se explicará con más detalle al comienzo del apartado 5.1.2.

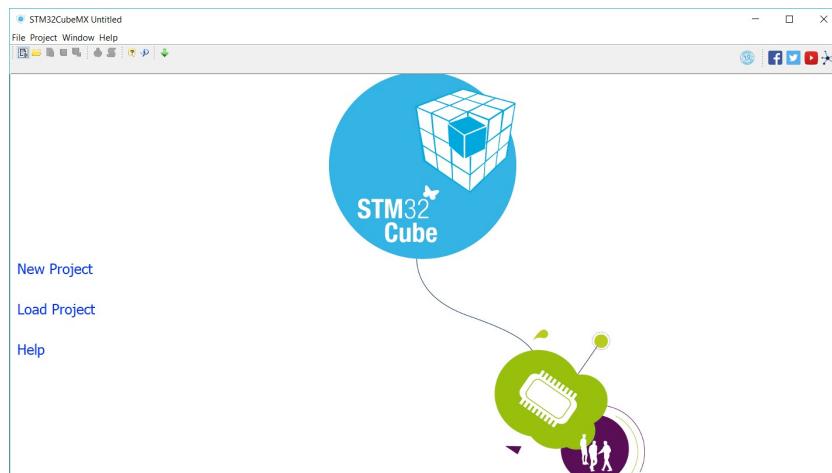


Figura 5.3: Inicializador de proyectos STM32CubeMX

La herramienta **STM32CubeMX** incluye una **base de datos de todos los microcontroladores disponibles**, permitiendo seleccionar el formato, encapsulado e incluso si se va a utilizar en su formato nucleo o en un kit de desarrollo. Incluye también documentación sobre cada microcontrolador y enlaces a distribuidores en caso de que el usuario final quiera comprarlo.

Asignación de funciones

Tras seleccionar el microcontrolador, se presenta al usuario una **zona de trabajo dividida en cuatro pestanas**, cada una destinada a configurar un conjunto de características del microcontrolador: **pines y sus funciones, reloj, interfaces de comunicación y consumo de energía**.

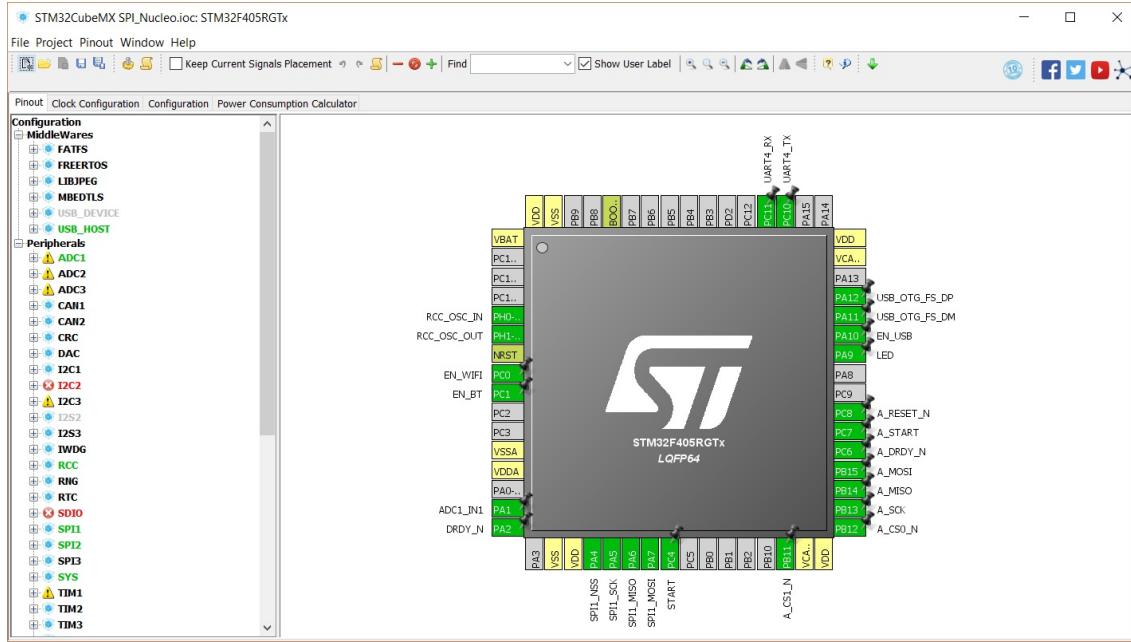


Figura 5.4: Espacio de trabajo del STM32CubeMX

La configuración de pines permite, no sólo ver de forma visual cada una de las funciones alternativas de cada pin, sino que además indica si alguna no está disponible y las alternativas en caso de existir.

En la figura 5.4 aparece la configuración utilizada en este proyecto. En la imagen se puede apreciar que todas las interfaces necesarias ya han sido asociadas a su pin correspondiente y aún quedan libres casi la mitad de los pines del total disponible. Esto es un indicador de la versatilidad de este dispositivo.

Configuración de los relojes

En este apartado se explicará la configuración de los relojes del sistema. Se trata de un punto muy importante puesto que un reloj mal configurado puede provocar errores en la comunicación, una mala gestión del tiempo e incluso que el propio microcontrolador no arranque.

La interfaz desarrollada por STM muestra de forma visual todos los relojes que intervienen en el dispositivo así como las relaciones que hay entre ellos. De esta forma basta con seguir las líneas que conectan los distintos tipos de reloj para saber las dependencias que existen y los resultados que obtendrán en función de los valores escogidos.

En caso de que alguna configuración sea incorrecta la propia herramienta está preparada para ofrecer una solución que se acerque lo máximo posible a los resultados deseados.

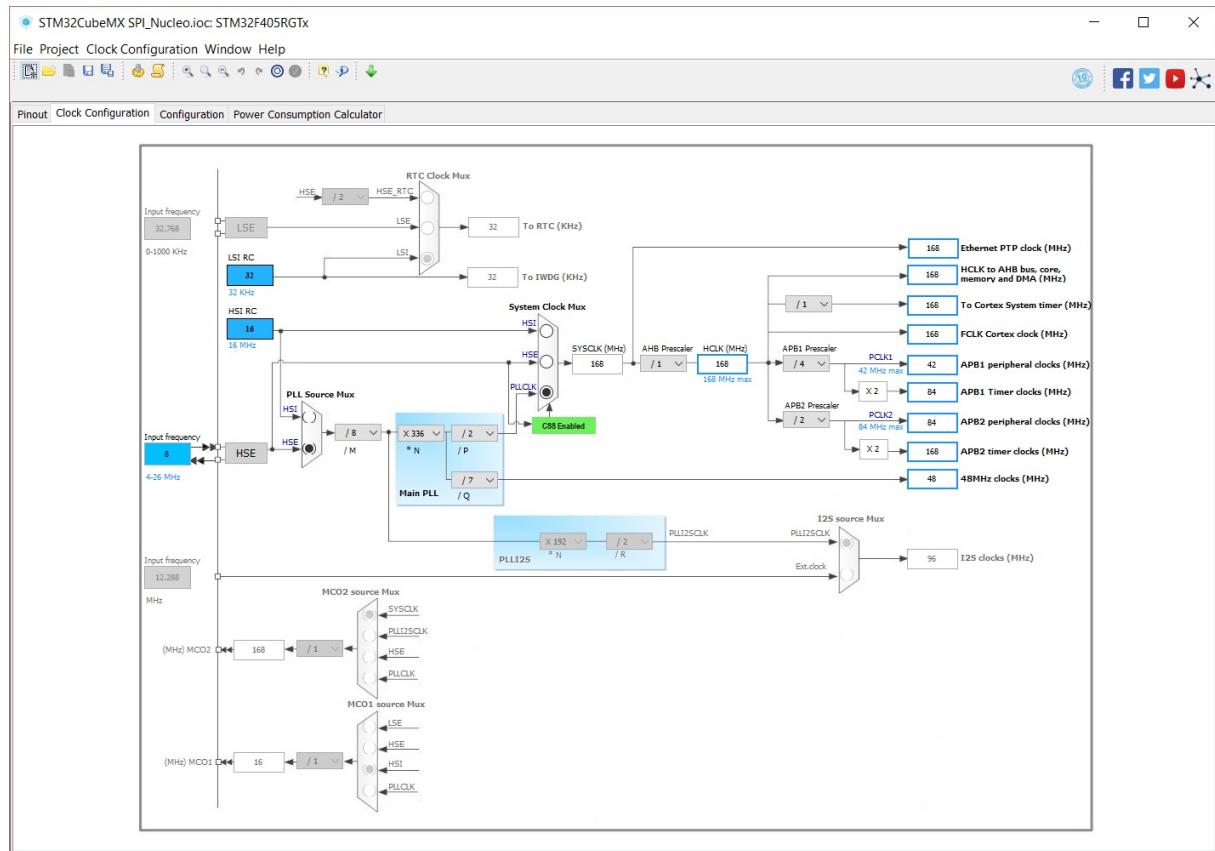


Figura 5.5: Configuración del reloj del microcontrolador

En la figura 5.5 representa la configuración aplicada al microcontrolador durante la realización de este proyecto. A la izquierda se muestran distintas fuentes disponibles mientras que a la derecha se ve la frecuencia final de reloj resultante en cada una de las partes del microcontrolador (periféricos, temporizadores, memoria, etc).

Como se puede ver, aprovecha ambos osciladores, el externo (HSE = 8MHz) y el interno (LSI = 16MHz) para, usando PLL, generar un reloj de una frecuencia mucho más elevada (**168 MHz**). Dicha frecuencia es, según el fabricante, la **máxima frecuencia alcanzable por el dispositivo**.

La función CSS garantiza que, en caso de fallo del reloj del microcontrolador, se genere una alerta y este entre en modo seguro. Esta función es muy útil cuando se está trabajando con elementos en los que la seguridad de las personas depende directamente del correcto comportamiento microcontrolador.

Configuración de los periféricos

La tercera pestaña permite **configurar los periféricos**, esto incluye interfaces de comunicación, puertos GPIO, USB, etc.

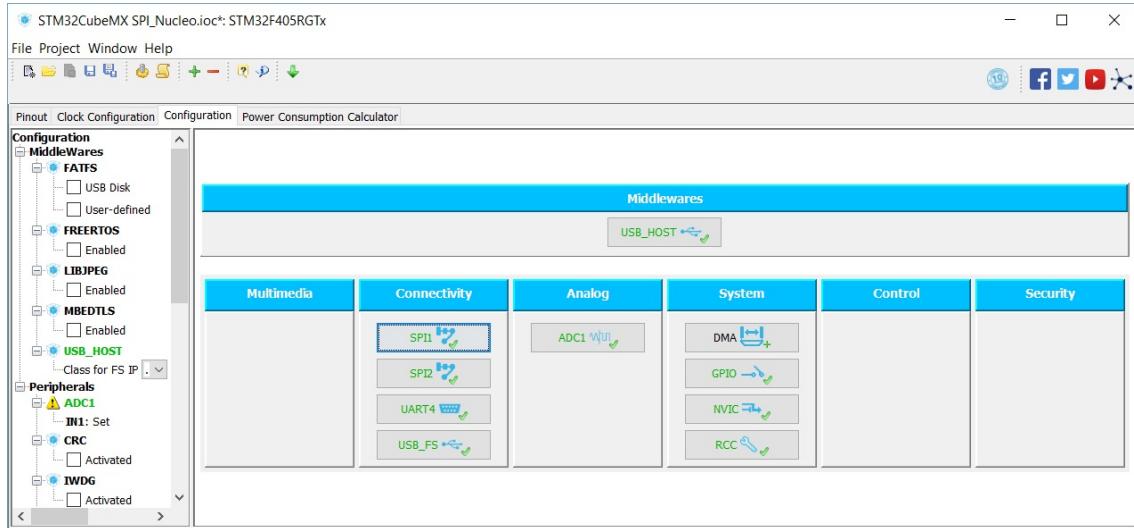


Figura 5.6: Configuración de las interfaces de comunicación

En función del estado del periférico este aparecerá marcado con un *tick* verde indicando que todo está correctamente configurado o una cruz roja avisando que alguna característica podría no estar disponible. Además, será posible modificar la mayoría de las opciones relacionadas con ese periférico de forma intuitiva.

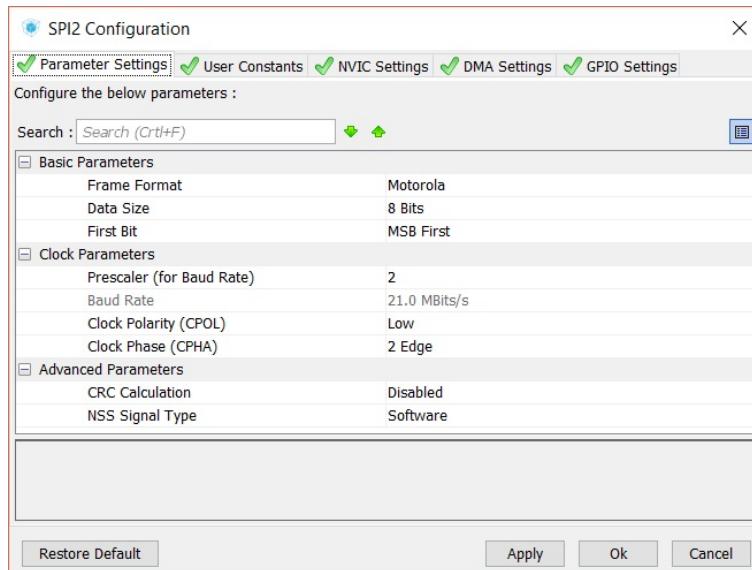


Figura 5.7: Detalle de la configuración de las interfaces de comunicación

En la figura 5.7 se muestran las opciones seleccionadas para el SPI que se comunicará con los ADS. Se puede cambiar la velocidad, el modo de transmisión e incluso si la gestión del pin \overline{CS} se realizará de forma automática o manual.

Cálculo de consumo

La última pestaña, “*Power Consumption Calculator*”, contiene numerosas opciones para el **análisis del consumo del microcontrolador**. En esta se puede estimar la **duración de la batería** en función de parámetros como el modo funcionamiento, el tipo de fuente de alimentación e incluso la temperatura.

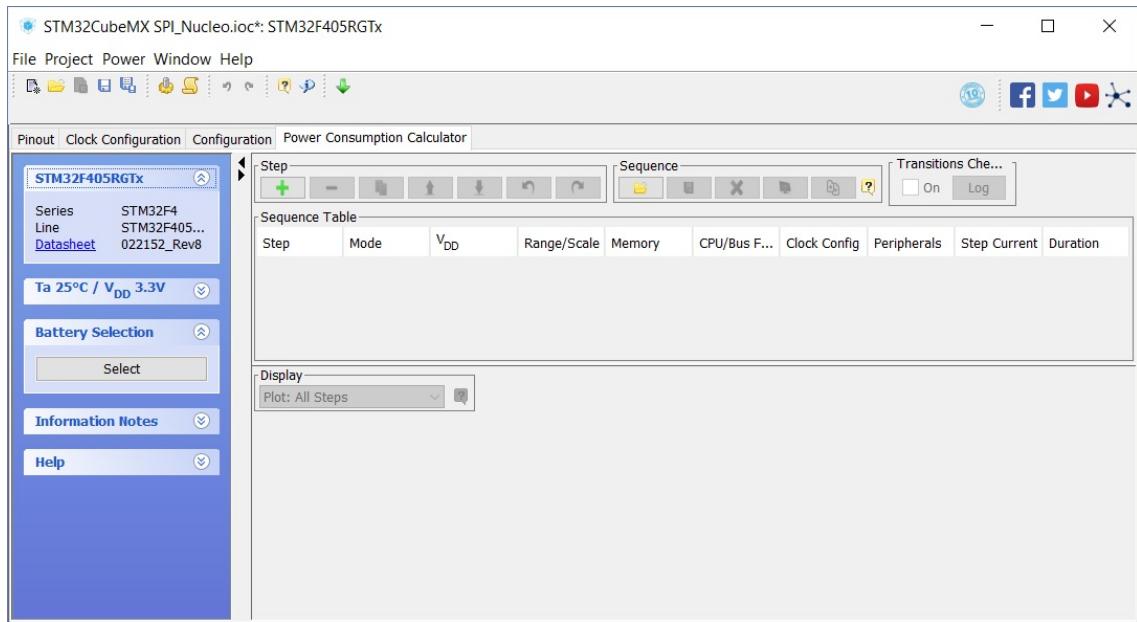


Figura 5.8: Calculadora del consumo del microcontrolador

Generar código

Finalmente se seleccionará el **IDE** correspondiente para el cual el STM32CubeMX debe generar la base de código sobre la que se construirá el proyecto.

Esta herramienta es capaz de crear un código base para casi cualquier IDE, pero de entre las opciones disponibles, **armKeil** es la herramienta que **presenta unos mejores resultados**. Dicho software crea el **código base** con la configuración seleccionada desde la interfaz y lo integra dentro de un proyecto ya configurado y listo para trabajar.

Es importante destacar que la utilización de STM32CubeMX no sólo está pensada para facilitar el comienzo del diseño proporcionando una base sobre la que trabajar, sino que también permite una **mayor portabilidad** del código ya que, si se siguen las reglas de programación sugeridas por la aplicación, esta permite realizar cambios de la configuración de los pines, los periféricos e incluso de familia de microcontrolador, todo ello **sin necesidad de reestructurar el código**.

El código que se ejecuta en el STM está dividido en **dos fases**. La primera, denominada comúnmente *setup*, sólo se ejecuta la primera vez que se enciende el microcontrolador. En esta se suelen declarar todas las variables globales, inicializar los periféricos y, en definitiva, preparar el microcontrolador para funcionar.

La segunda fase, denominada *loop* es, como su nombre indica, un **bucle infinito**. En esta fase se ejecuta todo el código de forma cíclica. Normalmente en esta fase se incluye el funcionamiento normal del sistema, utilizando condicionales para forzar el comportamiento esperado.

Al generar el proyecto, el STM32CubeMX estructura el código tal como se muestra en el código 5.1, dejando al usuario ciertas zonas libres para escribir en ellas y restringiendo la escritura en otras.

```
0  /* USER CODE BEGIN 1 */

/* USER CODE END 1 */

5  /* MCU Configuration—————*/  

   /* Reset of all peripherals , Initializes the Flash interface and the  

    Systick . */  

HAL_Init();

10 /* USER CODE BEGIN Init */

15 /* USER CODE END Init */
```

Código 5.1: Ejemplo de generación de código automáticamente

Respetar dichas restricciones es opcional aunque una vez modificadas no se podrá cambiar a la configuración original del microcontrolador usando STM32CubeMX, pues dicha herramienta las utiliza como referencia y no respetarlas puede ocasionar la pérdida del trabajo.

Por simplicidad, comodidad y compatibilidad, durante la realización de este trabajo **se han respetado dichas restricciones** construyendo todo el código en las zonas habilitadas para ello.

5.1.2. Implementación del firmware

A lo largo de las siguientes páginas se presentará una introducción a los elementos software utilizados, haciendo especial hincapié en aquel código cuya función es indispensable para cumplir con las especificaciones fijadas al comienzo del proyecto.

HAL

La programación de microcontroladores basados en Arduino y su IDE resulta especialmente intuitiva ya que desde sus comienzos ese era uno de los objetivos: acercar el mundo analógico y digital a gente poco versada en ello.

Los **ARM** de STM en cambio, están destinados a personas con cierta **experiencia programando microcontroladores**. Permiten libertad total proporcionando acceso a los registros sin restricciones y un control del hardware muy preciso. Si bien esta aproximación aumenta las posibilidades de la plataforma, también limita al número de personas que pueden ponerse a programar en ella así como la portabilidad de código. Cuanto más cercano sea al hardware mayor exclusivo será y, por lo tanto, se requerirán más cambios para portar dicho código a otra familia si en el futuro las especificaciones varían o se desean mejorar las características del sistema.

Con esta problemática en mente los desarrolladores de STM crearon un conjunto de herramientas y drivers que reciben el nombre de **Hardware Abstraction Layer (HAL)**.

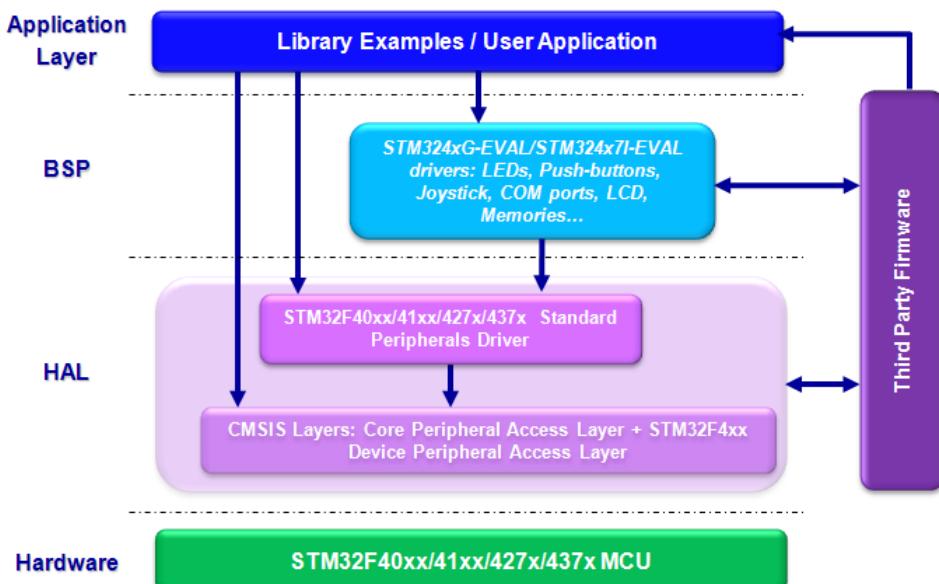


Figura 5.9: Resumen del funcionamiento y características del HAL [19]

La utilización de HAL permite tanto portar código entre familias de STM como simplificar sensiblemente el desarrollo de un proyecto, pues integra un gran número de funciones relacionadas con la gestión de los distintos puertos de comunicaciones y GPIO así como otras encargadas de realizar **operaciones matemáticas complejas**.

Estructura del código

El código está organizado siguiendo el esquema típico de una **máquina de estados**. Haciendo uso de condicionales de tipo “switch - case” se ha construído el programa representado en la figura 5.10.

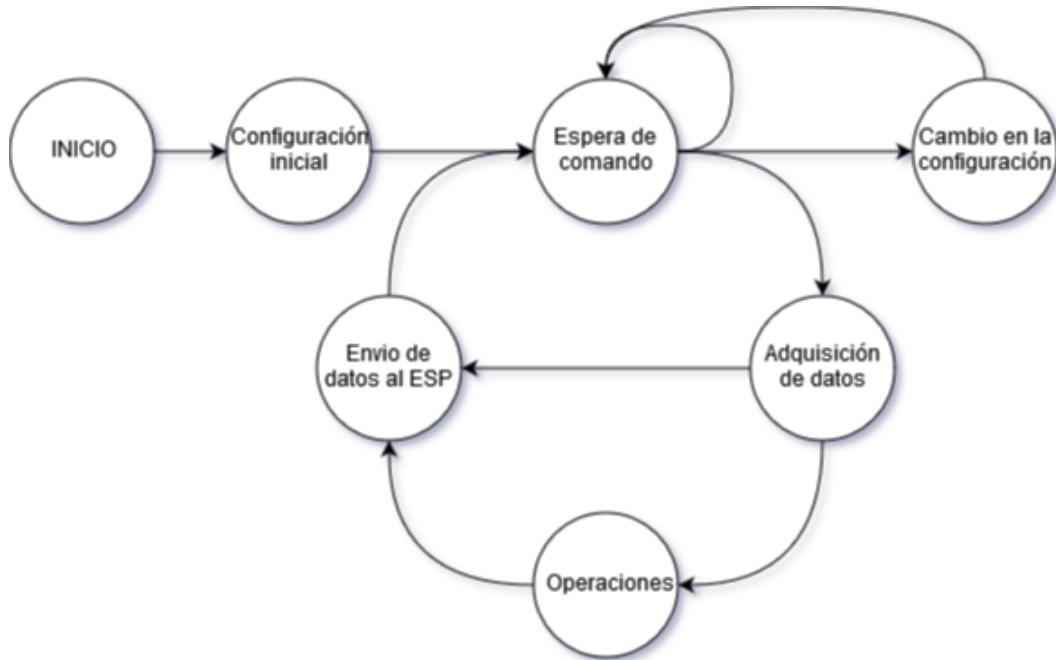


Figura 5.10: Estructura del programa

En primer lugar se realiza una **inicialización** de todas las variables, los periféricos y el resto de dispositivos que se encuentran conectados con el STM.

A continuación, el **microcontrolador** se queda **a la espera de un comando por parte del ESP-12E** que le indicará que hacer. Por el momento se han implementado **tres alternativas**, dos relacionadas con la adquisición de datos y una para cambios de configuración, pero está preparado para aumentar el número de condiciones con facilidad. De esta forma el software puede aumentar en funcionalidades sin que haya que reestructurar el código.

Si el microcontrolador recibe la orden de **adquirir datos**, este activa la comunicación SPI y queda a la espera de una confirmación por parte del ADS de que los datos están listos para ser leídos del bus.

Tras leer los datos el microcontrolador convierte la información a formato *float* y calcula la equivalencia en voltaje de la medida.

Posteriormente, en función de la orden recibida desde el ESP, el STM **transmite la información al ESP** o bien **realiza operaciones matemáticas sobre los datos** y finalmente los transmite al ESP. Estas operaciones pueden ser sumas, restas e incluso un filtrado de la señal para eliminar componentes indeseadas.

Transmisión de datos por SPI

Aunque la capa de abstracción del hardware facilita notablemente el trabajo de desarrollo, para poder trabajar con un bus de comunicaciones de forma eficiente y sin errores es necesario saber como funciona a nivel hardware y, desde ahí, realizar una abstracción progresiva.

En primer lugar será importante definir que elemento hará las **funciones de Máster** y **cual de Esclavo**, pues esto limitará las características de la comunicación y forzará la configuración de los GPIO de una forma u otra.

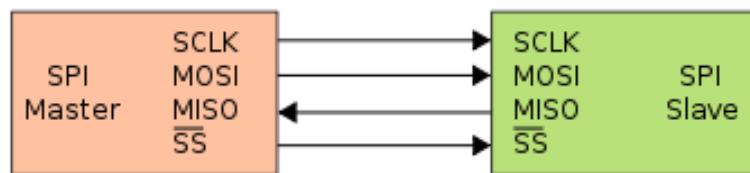


Figura 5.11: Esquema de comunicaciones SPI [2]

Para aprovechar la tasa de transferencia del SPI del STM al máximo lo ideal sería que fuese este el Máster en ambas comunicaciones (hacia el ADS y hacia el ESP). Esto es así porque la máxima velocidad de transmisión es definida por el Máster mientras que el Esclavo se adapta a las normas impuestas, siempre dentro de unos rangos establecidos por el fabricante.

Por desgracia, las limitaciones de las librerías disponibles para el **ESP-12E** fuerzan que este sea obligatoriamente el **Máster** de esa comunicación limitando la velocidad a la máxima que sea capaz de alcanzar el ESP.

Para la **transmisión de datos desde el STM** es muy importante tener en cuenta que el microcontrolador trabaja a bajo nivel. Las funciones implementadas en los *drivers* y librerías de abstracción de hardware facilitan el trabajo, pero debe tenerse presente en todo momento que se está transmitiendo información de registros de memoria y, por lo tanto, no se debe perder de vista el **tamaño** del dato ni el **tipo** del mismo.

Las funciones encargadas de transmitir información a través del puerto **SPI** son:

- HAL_StatusTypeDef HAL_SPI_Transmit
- HAL_StatusTypeDef HAL_SPI_Receive
- HAL_StatusTypeDef HAL_SPI_TransmitReceive

Las tres funciones necesitan el puntero al puerto por el que se transmitirá la información, la dirección de memoria de la misma, el tamaño del conjunto de datos y el tiempo de espera por cada transmisión.

El código 5.11 muestra la **cabecera de la función** encargada de **transmitir y recibir información** de forma simultánea. En él se puede ver con claridad el tipo de datos que necesita para realizar una transmisión satisfactoria.

```

0 HAL_StatusTypeDef HAL_SPI_TransmitReceive(
5   SPI_HandleTypeDef *hspi ,
  uint8_t *pTxData ,
  uint8_t *pRxData ,
  uint16_t Size ,
  uint32_t Timeout )

```

Código 5.2: Transmisión de datos a través de SPI con el STM

Las funciones encargadas de transmitir y recibir información, ya sea a través de SPI o a través de otro periférico, actúan de forma similar y necesitan las mismas variables. A pesar de que la función simplifica sensiblemente la gestión de memoria y pines, aún es necesario tener en cuenta ciertas limitaciones que marcarán el modo de trabajo:

- **Tamaño de transmisión fijo**

El tamaño de los datos a transmitir debe ser de 8 bits. En caso de querer transmitir una variable cuyo tamaño en memoria sea superior será necesario realizar una transformación.

- **Gestión de pines manual o automática**

La gestión de pines como “DRDY” y “START” se debe realizar de forma manual. “CS”, en cambio, está pensado para ser gestionado de forma manual o automática en función de las circunstancias.

- **Gestión de memoria**

El STM es capaz de transmitir una gran cantidad de datos por SPI. Es importante tener claro la cantidad de datos a recibir o transmitir, pues equivocarse puede suponer acceder a una dirección de memoria no permitida, recibir valores incorrectos o incluso corromper el código y forzar el reinicio del microcontrolador.

Al ser obligatorio transmitir datos de 8bits, se vuelve indispensable crear algún método para poder dividir aquellas variables cuyos tamaño sea superior (float, long, etc). El tipo de dato **union** definido en C está pensado justo para este cometido. Permite definir varias variables cuyo tamaño total sea equivalente y asignar la misma dirección de memoria a todas ellas. A continuación se muestra un fragmento del código necesario para dividir una variable **long** (32bits) en un array de 4 posiciones de tipo **int** (8bits cada una).

```

0 union miDato{
5   struct
  {
    uint8_t b[4];      // Array de bytes de tamaño equivalente
  } split;
  long dato;
} long_data;

```

Código 5.3: División de variables en otras de menor tamaño

Comunicación con el ADS

El **ADS** está preparado para recibir y transmitir información a través del puerto SPI integrado. Aunque puede transmitir información, el integrado sólo es capaz de actuar en **modo Esclavo** y **Full Duplex**.

La memoria del ADS está organizada en distintos registros, cada uno de los cuales tiene una función definida en la hoja de características del componente. La **configuración** del dispositivo se realiza de forma casi exclusiva mediante SPI **escribiendo directamente sobre el registro deseado**.

Al tratarse de registros de memoria, para su modificación resulta indispensable conocer su posición exacta. Aunque para una máquina esto no es un problema, para un ser humano puede resultar complicado recordar e identificar de forma eficiente cada uno de ellos. Para que la asociación sea más sencilla se ha creado un archivo ".h" con definiciones de todos los registros, asociando cada uno de ellos a un comando. Un ejemplo que ilustra esta problemática sería la modificación de los GPIO del ADS. Para poder realizar esta acción sin asociaciones sería necesario conocer que la dirección de memoria asociada al GPIO es la 0x14 mientras que tras la asociación basta con escribir en el registro GPIO y el microcontrolador se encargará de traducir este símbolo a su valor correspondiente. Con esta transformación se consigue además **aumentar la legibilidad del código**.

```
0 // Position in memory of registers: (Pg. 39)
1
2 #define ID          0x00  // (0000 0000)
3 #define CONFIG1     0x01  // (0000 0001)
4 #define CONFIG2     0x02  // (0000 0010)
5 .
6 #define CONFIG4     0x17  // (0001 0111)
```

Código 5.4: Ejemplo de definiciones de los registros y funciones del ADS

El manual de funcionamiento del ADS muestra distintas formas de configuración. Es posible **leer y/o escribir un registro** o varios de forma simultánea con pocos comandos. De acuerdo al manual, para realizar una escritura de un registro es necesario transmitir **tres datos al ADS**. El primero indicará la **dirección de memoria** sobre la que se trabajará. El segundo sirve para seleccionar el **número de registros** a escribir. Por último el tercer dato es el que se almacenará en memoria.

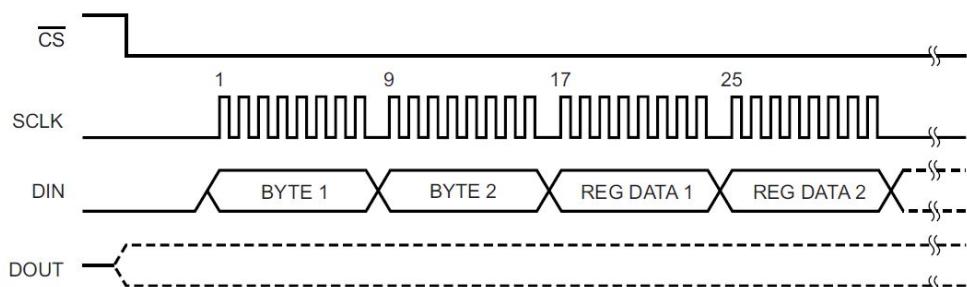


Figura 5.12: Escritura de dos registros del ADS [7]

El código generado en el STM para realizar esta función es el presentado en 5.5.

```

0 void adc_wreg(uint8_t reg, uint8_t val, SPI_HandleTypeDef *SPI) {
    uint8_t zero_t = 0x00;
    uint8_t zero_r = 0x00;
    uint8_t reg_temp = WREG | reg;
    uint8_t val_temp = val;

5    HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_RESET);
    HAL_SPI_TransmitReceive(SPI, &reg_temp, &zero_r, 1, 100);
    HAL_SPI_TransmitReceive(SPI, &zero_t, &zero_r, 1, 100);
    HAL_SPI_TransmitReceive(SPI, &val_temp, &zero_r, 1, 100);

10   HAL_Delay(1);
    HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_SET);
}

```

Código 5.5: Escritura en registros del ADS

La **lectura de los registros del ADS** se realiza de una forma análoga. En esta ocasión se transmiten los mismos dos datos que para escribir en un registro siendo la única diferencia que el tercer dato (y los siguientes en caso de ser más de un registro) se transmitirán desde el ADS hacia el STM tal y como aparece en la figura 5.13.

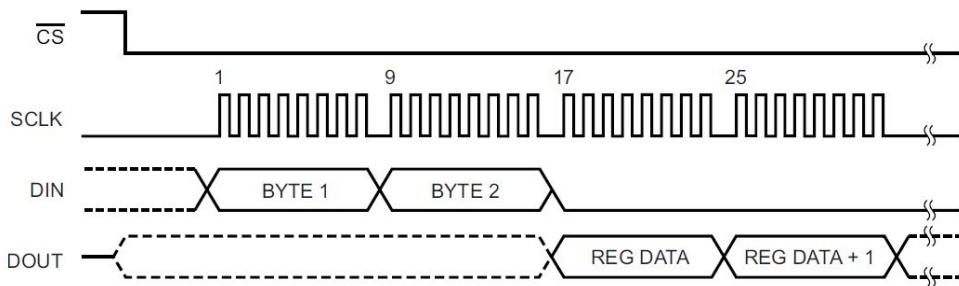


Figura 5.13: Lectura de dos registros del ADS [7]

```

0 uint8_t adc_rreg(uint8_t reg, SPI_HandleTypeDef *SPI) {
    uint8_t val = 0x00;
    uint8_t zero_t = 0x00;
    uint8_t zero_r = 0x00;
    uint8_t temp = RREG | reg;

5    HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_RESET);
    HAL_SPI_TransmitReceive(SPI, &temp, &zero_r, 1, 100);
    HAL_SPI_TransmitReceive(SPI, &zero_t, &zero_r, 1, 100);
    HAL_SPI_TransmitReceive(SPI, &zero_t, &val, 1, 100);

10   HAL_Delay(1);
    HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_SET);

15   return val;
}

```

Código 5.6: Lectura de registros del ADS

Para que la transmisión sea correcta es muy importante que todas las variables definidas sean independientes. Además, el ADS necesita que durante la lectura de datos el pin MOSI se mantenga en un 0 lógico todo el rato.

Lectura y conversión de datos

El ADS está construido utilizando un sistema adquisidor de 8 canales, todos adquiridos de forma simultánea utilizando un sistema $\Delta\Sigma$ de 24 bits y ganancia variable.

Tras adquirir los datos estos son codificados siguiendo el siguiente formato:

Señal de entrada, V_{IN} (INxP - INxN)	Código de salida ideal
+FS	7FFFFFFh
+FS/(2 ²³ - 1)	000001h
0	000000h
-FS	FFFFFFFFFFh
+FS(2 ²³ /(2 ²³ - 1))	800000h

Tabla 5.1: Equivalencia de voltaje ideal de entrada y código de salida

La tabla anterior es una modificación de la proporcionada en la hoja de características del ADS corrigiendo algunas pequeñas erratas. La variable “FS” corresponde a la escala completa de entrada del ADS. En este caso son 4.5V divididos por la ganancia (G).

Para **voltajes positivos** la decodificación de estos valores es muy sencilla, pues basta con aplicar una simple **regla de tres** en la que cada bit es un incremento de $4.5V/(2^{23}*G)$. Para **voltajes negativos** basta con **invertir** todos los bits y aplicar la misma regla de tres.

El bit más significativo indica si el voltaje final será positivo (0) o negativo (1) y es el que se ha usado como referencia para aplicar la conversión anterior en caso de ser necesaria.

Al trabajar con datos de 24 bits no es posible realizar una inversión directa del valor de la variable o los 8 bits más significativos se verían invertidos también. Por suerte el ADS transmite los datos en paquetes de 8 bits lo cual facilita tratarlos de forma independiente y, tras hacer la transformación a nivel de bit, crear una variable que represente el valor real.

Unir los tres bytes una vez se ha realizado la conversión es posible de múltiples formas. Por motivos de eficiencia y de reutilización del código se ha realizado la misma aproximación ya expuesta previamente haciendo uso de datos de tipo **union**.

El código 5.7, incluído en una librería creada para interactuar con el ADS, es el utilizado para este fin.

```
0 float32_t byte2float (uint8_t data_23_16, uint8_t data_15_8, uint8_t
    data_7_0, uint8_t ganancia)
{
    float32_t value = 0;

    union miDatos{
        struct
        {
            uint8_t b[4]; // Array de bytes de tamaño igual al tamaño de la
        primera variable: int = 2 bytes, float = 4 bytes
        }split;
        long dato;
    } long_data;
10 long_data.dato = 0;

    if (data_23_16>=0x80) //1000 0000
15 {
    long_data.split.b[2] = ~data_23_16;
    long_data.split.b[1] = ~data_15_8;
    long_data.split.b[0] = ~data_7_0;
    value = long_data.dato;
20    value = -value;
}
else
{
    long_data.split.b[2] = data_23_16;
25    long_data.split.b[1] = data_15_8;
    long_data.split.b[0] = data_7_0;
    value = long_data.dato;
}
30 value = value*4.5f/(8388607.0f*ganancia); //4.5/(2^23*G)

    return value;
}
```

Código 5.7: Decodificación de datos binarios a Voltajes

El código anterior realiza una conversión triple: 3 bytes \Rightarrow long \Rightarrow float, donde el valor final devuelto es el voltaje real medida una vez se ha aplicado el factor de ganancia correspondiente.

El ADS tiene embebido un sistema de control basado en comandos a través de SPI. Estos comandos permiten realizar ciertas operaciones en tiempo real que facilitan el control del integrado y amplian sus características.

La tabla 5.2, sacada de la hoja de características del componente, muestra todos los comandos que el ADS es capaz de reconocer, sus funciones y el conjunto de bytes que representa cada uno. Los últimos dos comandos corresponden a la lectura y escritura en registros y ya han sido explicados previamente.

COMMAND	DESCRIPTION	FIRST BYTE	SECOND BYTE
System Commands			
WAKEUP	Wake-up from standby mode	0000 0010 (02h)	
STANDBY	Enter standby mode	0000 0100 (04h)	
RESET	Reset the device	0000 0110 (06h)	
START	Start and restart (synchronize) conversions	0000 1000 (08h)	
STOP	Stop conversion	0000 1010 (0Ah)	
Data Read Commands			
RDATA C	Enable Read Data Continuous mode. This mode is the default mode at power-up. ⁽¹⁾	0001 0000 (10h)	
SDATAC	Stop Read Data Continuously mode	0001 0001 (11h)	
RDATA	Read data by command; supports multiple read back.	0001 0010 (12h)	
Register Read Commands			
RREG	Read <i>n nnnn</i> registers starting at address <i>r rrrr</i>	001 <i>r rrrr</i> (2xh) ⁽²⁾	000 <i>n nnnn</i> ⁽²⁾
WREG	Write <i>n nnnn</i> registers starting at address <i>r rrrr</i>	010 <i>r rrrr</i> (4xh) ⁽²⁾	000 <i>n nnnn</i> ⁽²⁾

Tabla 5.2: Comandos del ADS1299

Durante la realización de este proyecto ha sido necesario utilizar la mayor parte de los comandos presentados en la tabla anterior. Cabe destacar que aquellos categorizados como “Comandos del sistema” han sido necesarios durante la secuencia de inicio del ADS casi en exclusiva, pues no se han implementado funciones de ahorro de energía. Estas se han contemplado como una mejora a llevar a cabo en futuras iteraciones de este proyecto.

El ADS cuenta con dos modos de lectura: continua y bajo demanda. Los comandos categorizados como “Comandos de lectura” están relacionados con la gestión de estos dos modos y se dividen a su vez en dos tipos según su función:

- **Configuración**

RDATA C y SDATAC activan y desactivan el modo de lectura continua respectivamente. El ADS se encuentra por defecto en modo continuo tras completar la secuencia de inicio.

- **Lectura**

RDATA solicita en el modo de lectura bajo demanda la transmisión de los datos capturados en ese momento.

El modo bajo demanda está pensado para aquellos sistemas en los que es necesario adquirir poca información a lo largo del tiempo. En esta configuración el ADS se mantiene inactivo hasta que recibe el comando de lectura y sólo entonces es cuando la realiza. Este comportamiento favorece la implementación de sistemas de ahorro de energía basados en los comandos STANDBY y WAKEUP. De esta forma es posible aumentar la autonomía del sistema completo considerablemente.

Por desgracia, este modo no es compatible con la transmisión de grandes cantidades de datos o mediciones muy continuas. Por cada medida es necesario utilizar el comando RDATA, afectando directamente al rendimiento y eficiencia en la transmisión.

La figura 5.14 ilustra la transmisión de información usando el modo bajo demanda.

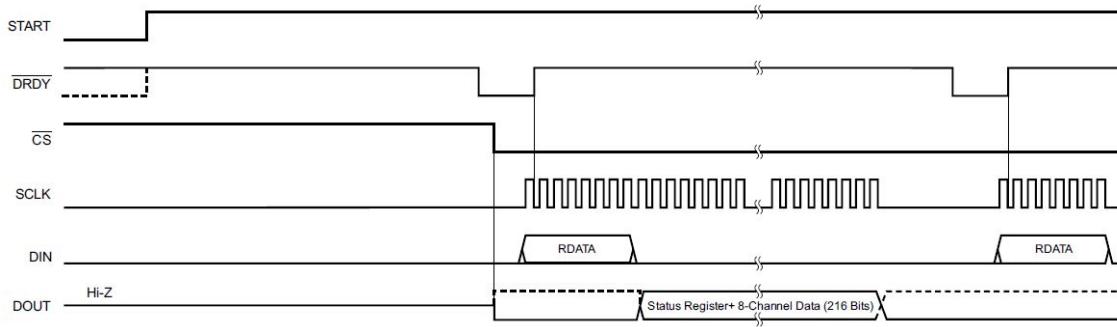


Figura 5.14: Modo de uso del comando RDATA

Por los motivos explicados anteriormente este modo no será utilizado normalmente pero aún así se ha decidido trabajar en su implementación en caso de que fuese necesario en el futuro. El código utilizado para adquirir datos es el siguiente:

```

0 void one_shot ( uint8_t data[] , SPI_HandleTypeDef *SPI)
{
    uint8_t zero = 0x00;
    uint8_t cmd = RDATA;

5     while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) == GPIO_PIN_SET)
        {}

    HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_RESET);

    HAL_SPI_TransmitReceive(SPI, &cmd, &zero, 1, 100);
10
    read_data_frame(data, SPI);

    while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==
        GPIO_PIN_RESET) {}

15
    HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_SET);
    update_bias_ref(data, SPI);
}

```

Código 5.8: Lectura de datos en modo *one-shot*

Para recibir una gran cantidad de información de forma ininterrumpida el ADS cuenta con el modo continuo. Se transmite el comando RDATAC y desde ese momento el ADS empieza a adquirir información y transmitirla. El pin DRDY es utilizado por el ADS para indicar que la siguiente tanda de datos está disponible para su lectura.

La figura 5.15 representa la transmisión de información en modo continuo y el código 5.9 el código equivalente.

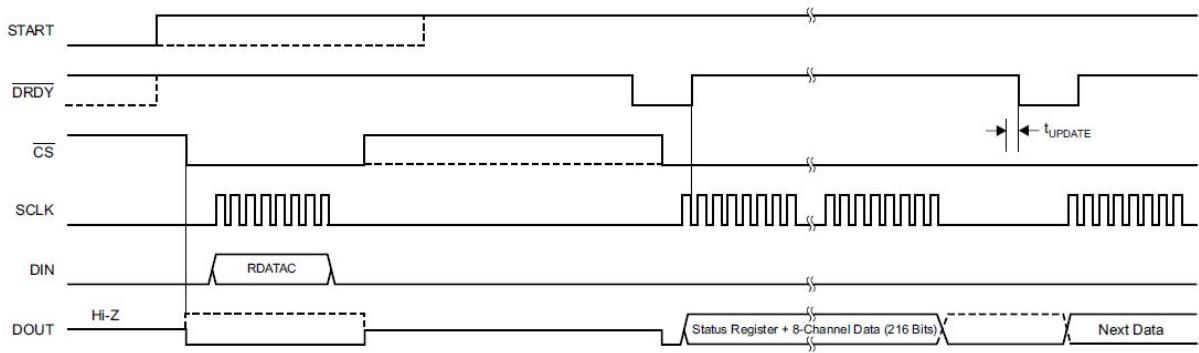


Figura 5.15: Transmisión de datos en modo continuo

```

0 void acquire_array_data ( uint8_t data[] , float32_t channel_1[] , float32_t
    channel_2[] , float32_t channel_3[] , float32_t channel_4[] , float32_t
    channel_5[] , float32_t channel_6[] , float32_t channel_7[] , float32_t
    channel_8[] , uint8_t gain[] , SPI_HandleTypeDef *SPI)
{
    int debug = 255;
    // read LENGTH_SAMPLES samples

5    adc_send_command(RDATAC, SPI);

    while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) == GPIO_PIN_SET)
    {}

    HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_RESET);
10   read_data_frame(data, SPI);

    channel_1[0]=byte2float (data[1*3] , data[1*3+1] , data[1*3+2] , gain[0]);
    channel_2[0]=byte2float (data[2*3] , data[2*3+1] , data[2*3+2] , gain[1]);
15   channel_3[0]=byte2float (data[3*3] , data[3*3+1] , data[3*3+2] , gain[2]);
    channel_4[0]=byte2float (data[4*3] , data[4*3+1] , data[4*3+2] , gain[3]);
    channel_5[0]=byte2float (data[5*3] , data[5*3+1] , data[5*3+2] , gain[4]);
    channel_6[0]=byte2float (data[6*3] , data[6*3+1] , data[6*3+2] , gain[5]);
    channel_7[0]=byte2float (data[7*3] , data[7*3+1] , data[7*3+2] , gain[6]);
20   channel_8[0]=byte2float (data[8*3] , data[8*3+1] , data[8*3+2] , gain[7]);

    while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==
        GPIO_PIN_RESET){}

    for (int i = 1; i<LENGTH_SAMPLES; i++)
25   {
        while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==
            GPIO_PIN_SET){}

        read_data_frame(data, SPI);

        channel_1[i]=byte2float (data[1*3] , data[1*3+1] , data[1*3+2] , gain[0]);
        channel_2[i]=byte2float (data[2*3] , data[2*3+1] , data[2*3+2] , gain[1]);
        channel_3[i]=byte2float (data[3*3] , data[3*3+1] , data[3*3+2] , gain[2]);
        channel_4[i]=byte2float (data[4*3] , data[4*3+1] , data[4*3+2] , gain[3]);
        channel_5[i]=byte2float (data[5*3] , data[5*3+1] , data[5*3+2] , gain[4]);
    }
}

```

```

35    channel_6[i]=byte2float (data[6*3], data[6*3+1], data[6*3+2], gain[5]);
      channel_7[i]=byte2float (data[7*3], data[7*3+1], data[7*3+2], gain[6]);
      channel_8[i]=byte2float (data[8*3], data[8*3+1], data[8*3+2], gain[7]);

      while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==
40      GPIO_PIN_RESET){}
}

      HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_SET);
}

```

Código 5.9: Lectura de datos en modo continuo

El STM no permite trabajar con vectores de varias dimensiones de la misma forma que otros lenguajes de programación, así que para capturar todos los canales ha sido necesario repetir código. A cambio, el escalado del código de un canal a varios se realiza de una forma sencilla e intuitiva. En posteriores revisiones de este proyecto se valorará su sustitución por otro sistema más eficiente.

DSP - Implementación de filtros

Una de las características que diferencian los microcontroladores de esta familia de los de otras es la **capacidad de realizar operaciones matemáticas** sin que ello repercuta de forma muy significativa en el rendimiento.

Para facilitar su utilización y unificar a todos los desarrolladores ARM ha creado una librería llamada CMSIS-DSP que engloba todas las funcionalidades de procesado de señal más comunes, optimizando dichos algoritmos para hacerlos lo más eficientes posible. Esta librería incluye los siguientes tipos de operaciones:

- Operaciones matemáticas básicas, rápidas y complejas
- **Filtrado**
- Matrices
- Transformaciones
- Estadísticas
- Controladores (PID, senos, etc)
- Interpolación
- Apoyo

Estas funciones utilizan variables de tipo enteros de 8, 16 y 32 bits y float de 32 bits. En esta ocasión se han utilizado exclusivamente las funciones de filtrado de señal, concretamente aquellas dedicadas a la realización de filtros FIR y su aplicación a señales.

ARM ha puesto a disposición de los consumidores una página web de referencia en la que se puede encontrar una descripción detallada de cada función incluyendo su modo de uso y ejemplos [20].

El filtrado de una señal con un filtro FIR en ARM es un proceso sencillo pero que engloba varias herramientas, pues el diseño del filtro se debe realizar de forma independiente. La función utilizada para este fin es “arm_fir_f32()”.

Esta función recibe una señal en formato **float32_t** ($x[n]$), los coeficientes del filtro y la longitud de la ventana y devuelve una señal filtrada ($y[n]$) con las características deseadas.



Figura 5.16: Diagrama de bloques del filtrado de una señal $x[n]$

El código necesario para aplicar un filtro FIR a una señal es el siguiente:

```

0  /* 
   ** Includes necesarios
   */
5 #include "arm_math.h"
#include "math_helper.h"
10 const float32_t firCoeffs32[NUM_TAPS] = {h1, h2, ..., hn}
/* 
 * Variables globales para el filtro FIR
 */
15 uint32_t blockSize = BLOCK_SIZE;
10 uint32_t numBlocks = TEST_LENGTH_SAMPLES/BLOCK_SIZE;
/* 
arm_fir_instance_f32 S;
arm_status status;
float32_t *inputF32, *outputF32;
15 /* Inicialización de buffers de entrada y salida */
inputF32 = &testInput_f32_1kHz_15kHz[0];
outputF32 = &testOutput[0];
/* Inicialización de la estructura de la instancia del filtro FIR. */
arm_fir_init_f32(&S, NUM_TAPS, (float32_t *)&firCoeffs32[0], &firStateF32
[0], blockSize);
20 /* 
** Aplicación del filtro a cada conjunto de blockSize
*/
25 for (i=0; i < numBlocks; i++)
{
    arm_fir_f32(&S, inputF32 + (i * blockSize), outputF32 + (i * blockSize),
}
    
```

Código 5.10: Filtro FIR en STM

Es una función muy versátil que permite, entre otras cosas, preparar un gran número de filtros y utilizar el más adecuado para cada ocasión sin apenas cambiar el código.

La utilización de filtros FIR en este proyecto está centrada en la eliminación de una de las componentes en frecuencia que más afecta a todas las señales: la de 50 Hz.

Para su eliminación, normalmente se aplica un filtro IIR denominado **Notch** o de rechazo de banda a la señal, pues estos presentan una selectividad en frecuencia muy alta a pesar de poder ser inestables.

Haciendo uso de la función **arm_fir_f32** es posible implementar una **aproximación** a dicho filtro con uno de tipo FIR cuyas características sea lo suficientemente buenas como para atenuar las componentes en frecuencia de 50Hz sin afectar demasiado al resto de la señal. Esto se consigue creando un filtro de rechazo de banda lo suficientemente estrecho centrado en torno a la frecuencia deseada a la par que se mantiene un rizado mínimo en el resto de las frecuencias.

El diseño del filtro se puede hacer con cualquiera de las herramientas disponibles en MatLab. La función “fir1” es capaz de definir un filtro con cualquier característica. Tras una primera aproximación con esa función se consiguió un filtro con la respuesta en frecuencia de la figura 5.17.

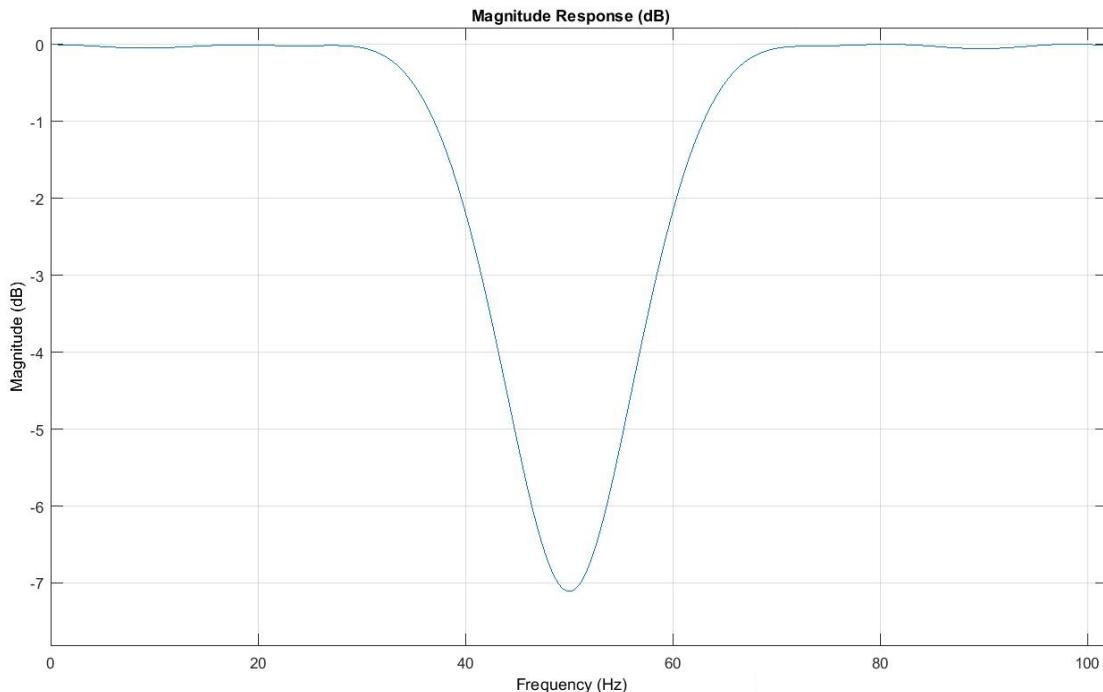


Figura 5.17: Respuesta en frecuencia del filtro definido con fir1

Este filtro presenta un comportamiento similar al esperado y un **rizado apenas apreciable**, pero con un factor de calidad y de rechazo de banda muy bajo.

Utilizando otra herramienta diseñada por F.S. Schlindwein disponible en el *workshop* de MatLab es posible ajustar aún más el filtro sin apenas esfuerzo.

Tras un par de iteraciones el resultado final es el representado en la figura 5.18.

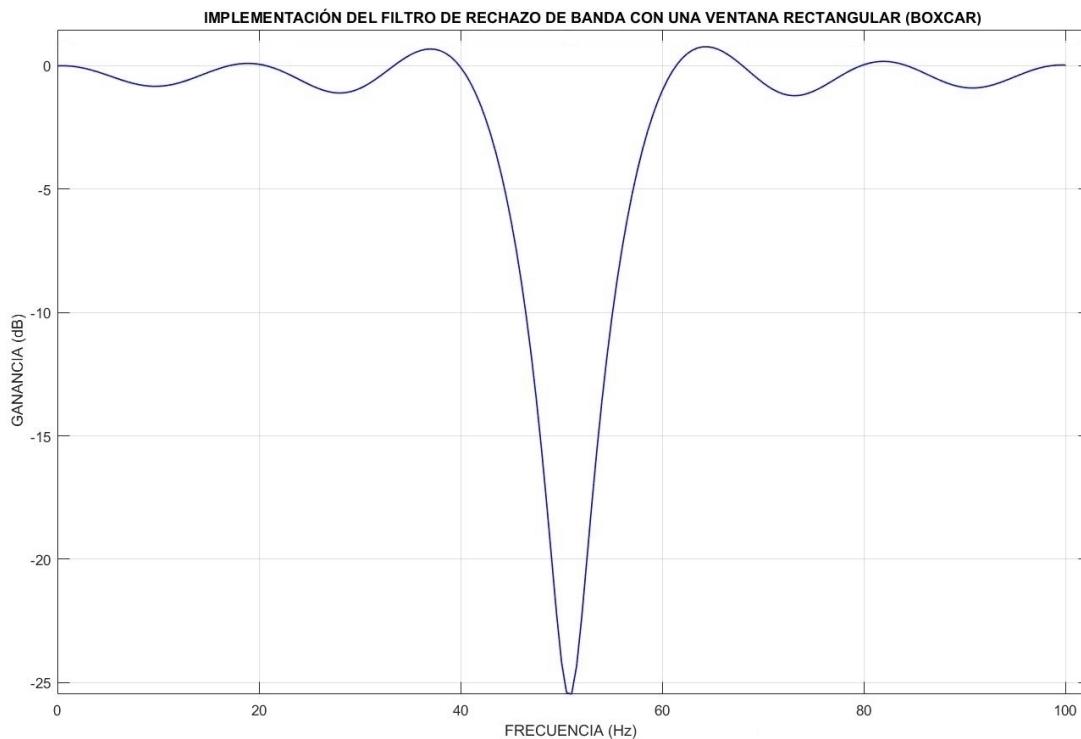


Figura 5.18: Respuesta en frecuencia del filtro FIR definido con FIRFILT [21]

Este filtro presenta un ancho de banda similar, pero a cambio de introducir un pequeño rizado en la banda deseada, la **atenuación a 50 Hz es sensiblemente mayor** que en el caso representado en la figura 5.17.

El filtrado analógico es inmediato pero muy poco flexible. Utilizando filtros digitales se consigue adaptar el filtro en función de las necesidades y, en caso necesario, siempre es posible eliminar dicho filtro simplemente cambiando el software. No se debe olvidar que desde el punto de vista del coste/cambio, un cambio a nivel de software casi siempre será mucho más **rentable** en el producto final que uno a nivel de hardware.

Por desgracia la utilización de esta librería provoca que el código supere las limitaciones de la versión gratuita de Keil haciendo necesaria una licencia para poder compilar.

Comunicación con la interfaz inalámbrica

La transmisión de datos a la interfaz inalámbrica se realiza de forma similar a la ya explicada para comunicar el STM con el ADS. En esta ocasión el **STM actuará como Esclavo** mientras que el dispositivo que actúa como Master es aquel que hace la función de transmisor de la información. Esto es así debido a las limitaciones impuestas por el ESP12-E.

En la comunicación con el otro microcontrolador hay **tres etapas** bien diferenciadas. En la primera la información transmitida son **comandos** que indican al STM el comportamiento que deberá seguir.

La segunda, centrada en la transmisión de pequeñas cantidades de información, no necesita una gran eficiencia en la transmisión. Esta información se utilizará por parte del STM para **configurar el ADS**.

La última fase contempla la transmisión de grandes cantidades de información y por tanto su optimización debe ser máxima.

El código 5.11 es el utilizado para transmitir información al ESP y recibirla en caso de que las restricciones de tiempo no sean críticas (primera y segunda fase).

```
0 uint8_t commandFromESP(uint8_t command, SPI_HandleTypeDef *hspi1)
{
    uint8_t response = 0x00;

    HAL_GPIO_WritePin(DRDY_N_GPIO_Port, DRDY_N_Pin, GPIO_PIN_RESET);
    HAL_SPI_TransmitReceive(hspi1, &command, &response, 1, 100);
    HAL_GPIO_WritePin(DRDY_N_GPIO_Port, DRDY_N_Pin, GPIO_PIN_SET);
    HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin); //LED OFF
    HAL_Delay(50);

10   return response;
}
```

Código 5.11: Transmisión de datos a través de SPI con el STM

Por la forma en la que está construida la transmisión de datos a través de SPI, la gestión de los pines involucrados directamente en la transmisión la información se realiza de forma automática y es tremadamente eficiente. Sin embargo, la gestión de los pines **CS** y **DRDY** se realiza de forma manual y la sincronización entre ambos microcontroladores provoca que la eficiencia de la transmisión se reduzca drásticamente.

El STM contempla la transmisión de grandes cantidades de información pero, para que esta funcione correctamente, dicha información debe estar en posiciones de memoria consecutivas. Aprovechando esta característica, es posible minimizar el retardo introducido en la sincronización de ambos microcontroladores creando un *buffer* de transmisión de un tamaño lo suficiente grande. De esta forma, cuanto **mayor sea la cantidad de datos a transmitir, mayor será la eficiencia** de la comunicación.

El código utilizado en la fase 3 es equivalente al código 5.11 siendo la única diferencia la cantidad de bytes a transmitir, que es adaptada en función al tamaño del *buffer* de transmisión.

Programación del microcontrolador

Finalmente, tras tener preparado el *firmware* del STM es el momento de cargar dicho código en el microcontrolador. Los microcontroladores de esta familia cuentan con dos métodos para cargar código: usando *jtag* y mediante un *bootloader* embebido.

La primera opción es muy interesante, pues además de programar el microcontrolador da acceso a otras funciones relacionadas con la depuración del código (ejecución en tiempo real, *breakpoints*, visualización de variables, etc). Sin embargo para funcionar es necesario **utilizar un hardware específico**, y al no disponer de él durante el comienzo del proyecto no se tuvo en cuenta su inclusión durante el desarrollo de la placa.

La segunda opción aprovecha los puertos de UART del microcontrolador para, usando un conversor UART ↔ USB, programar el STM. Este sistema sólo permite la programación del dispositivo, pero a cambio el **hardware extra necesario es mínimo**, más teniendo en cuenta que ya se disponía de dicho conversor y, por lo tanto, no fue necesario realizar el desembolso.

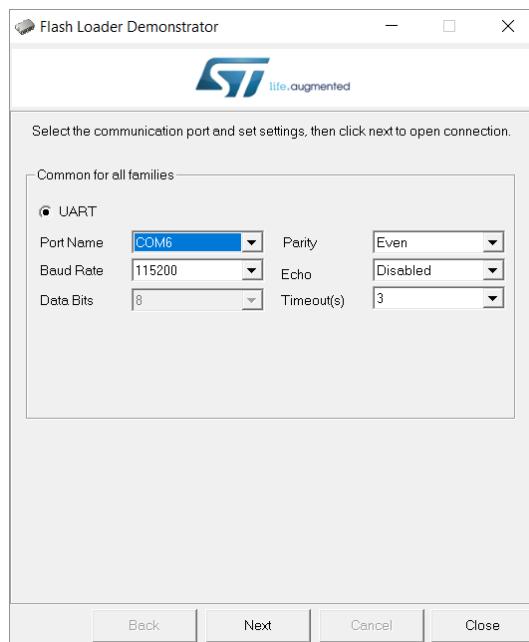


Figura 5.19: Herramienta de programación a través de UART

Para poder reprogramar el microcontrolador es necesario reiniciarlo mientras se cortocircuitan los pines 1 y 2 del conector preparado para ese fin.

5.2. Interfaz inalámbrica

Los dispositivos integrados en la interfaz inalámbrica han sido relegados a una función casi meramente de transmisión pero no se debe olvidar que ambos son microcontroladores y que, por tanto, los dos necesitan ejecutar un *firmware* para funcionar.

Estos dispositivos fueron seleccionados con múltiples objetivos en mente. Por un lado su relación características/coste es muy adecuada, permitiendo crear un sistema final muy completo con un coste muy bajo. Por otro, ambos microcontroladores son compatibles con el IDE de Arduino.

Al ser compatible con Arduino IDE se **minimiza el tiempo de desarrollo** del software, pues el propio IDE realiza la función de abstracción del hardware e incluye un gran número de ejemplos funcionales. Además, al ser ambos dispositivos compatibles, las modificaciones necesarias para **portar el código** de un microcontrolador a otro son mínimas. Por este motivo a lo largo de este capítulo se expandirá el código implementado en el ESP12-E, quedando el del Bluetooth en segundo plano.

5.2.1. Arduino

Arduino fue una iniciativa de Massimo Banzi en 2005 pensada para acercar la electrónica a los estudiantes de un modo sencillo y económico. Con el paso del tiempo esta plataforma ha ido creciendo, incorporando nuevas placas y microcontroladores, y mejorando tanto en la interfaz como en la comunidad que la respalda.



Figura 5.20: Arduino IDE utilizado para la programación del ESP

Por defecto Arduino incluye sólo aquellas placas comercializadas oficialmente por la empresa, pero los desarrolladores han habilitado la posibilidad de crear nuevas librerías e importarlas para que así cualquier persona con suficientes conocimientos pueda añadir las suyas propias.

5.2.2. Configuración

Para trabajar con el ESP12-E (esp8266) basta con importar la librería y escoger el componente en el selector de placas integrado.

Las placas no contempladas de forma oficial por Arduino se importan desde File ⇒ Preferences.

A continuación se debe introducir la ruta al .json de la librería para que así Arduino tenga un repositorio desde el que buscar aquellas placas que no están instaladas. El resultado final es el mostrado en la figura 5.21.

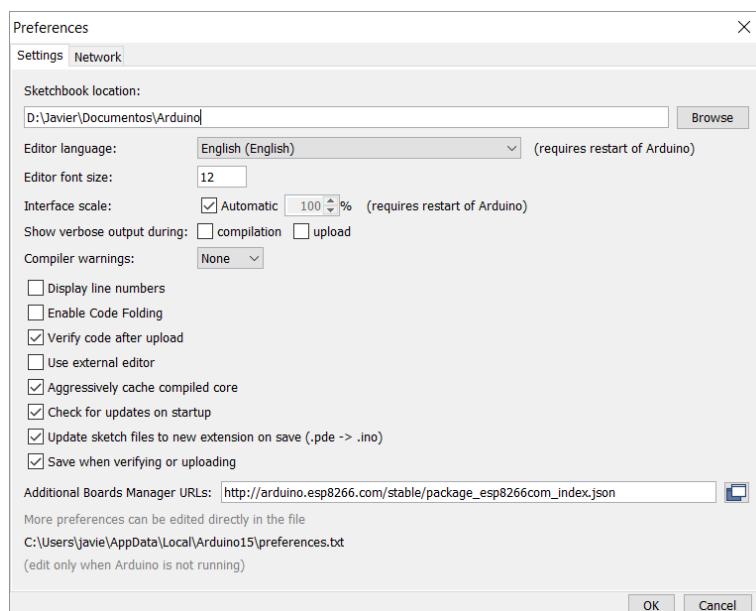


Figura 5.21: Importar librerías en Arduino

Por último se procede a la instalación de las placas deseadas. Para ello se ha habilitado un repositorio con buscador e interfaz gráfica. Este es accesible desde Tools ⇒ Board ⇒ Boards Manager...



Figura 5.22: Instalar librerías para nuevas placas

5.2.3. Firmware

En los siguientes apartados se detallará el software diseñado para funcionar dentro del microcontrolador.

Maquina de estados

El código de Arduino está organizado de forma similar al ya explicado para el STM. Este se divide en dos partes, una de *setup* que sólo se ejecuta una vez y otra cíclica denominada *loop*.

Para caracterizar el comportamiento del ESP en todo momento se ha creado un sistema de máquina de estados basada en estructuras *switch/case* similar al ya explicado para el STM.

En esta ocasión la variable de control principal viene determinada por el sistema de control externo y, en función del estado en el que se encuentre el microcontrolador, este interactúa de una forma u otra con el STM.

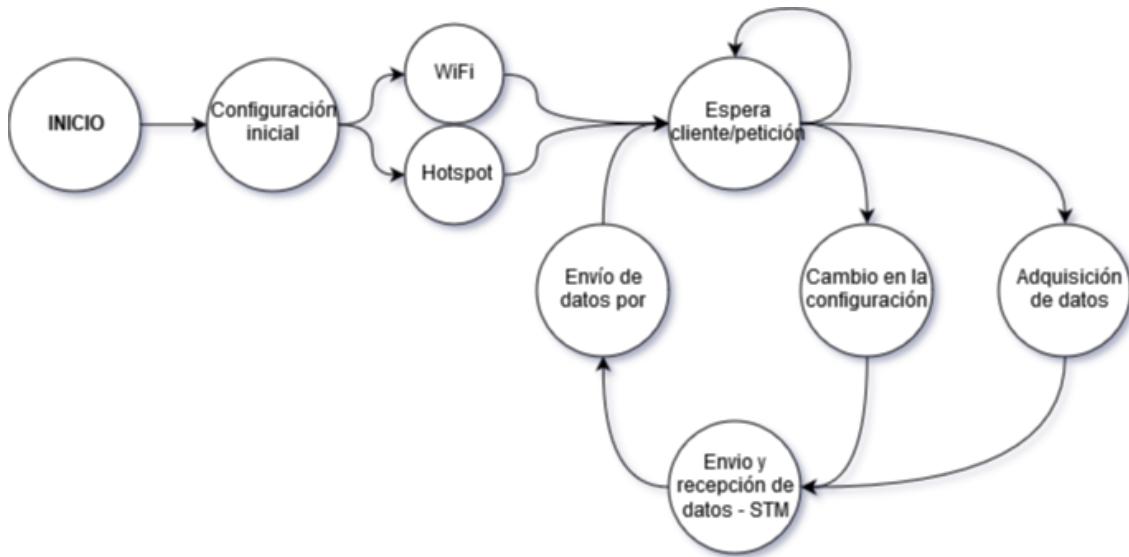


Figura 5.23: Maquina de estados del ESP

Cada una de las funciones deseadas para el sistema tiene su estado asociado. Además ha sido necesario implementar un sistema de sincronización entre el ESP y el STM que asegure que la transición entre estados de ambos microcontroladores sea siempre simultánea.

Transmisión de datos por SPI

La transmisión de datos a través de SPI es una de las primeras funciones que los desarrolladores incluyen en las librerías cuando las adaptan a un componente. La librería del ESP8266 no es una excepción y por lo tanto incluye bastantes funciones y definiciones como para poder trabajar de forma cómoda con el ESP en modo Master.

Para funcionar correctamente, eso sí, es necesario conocer perfectamente que pines serán utilizados para la transmisión y **configurar tanto los pines como el propio SPI** de acuerdo a las necesidades del proyecto.

El ESP cuenta con varios buses SPI. En este proyecto se hace uso de los pines 5 (HSPI_CLK) , 6 (HSPI_MISO) y 7 (HSPI_MOSI) y un GPIO adicional para el CS, en esta ocasión el pin 16.

El código 5.12 muestra una configuración funcional de todos los pines así como del SPI.

```

0 SPI.setClockDivider(SPI_CLOCK_DIV2); //Divides 16MHz clock by 2 to set CLK
    speed to 8MHz
1 SPI.setDataMode(SPI_MODE1); //clock polarity = 0; clock phase = 1 (pg. 8)
2 SPI.setBitOrder(MSBFIRST); //data format is MSB (pg. 25)

3 pinMode(HSPI_CS,OUTPUT);
4 pinMode(HSPI_CLK,SPECIAL);
5 pinMode(HSPI_MISO,SPECIAL);
6 pinMode(HSPI_MOSI,SPECIAL);
7 pinMode(DRDY_N,INPUT);
8 pinMode(START,OUTPUT);

```

Código 5.12: Configuración de pines para el uso de SPI

Por simplicidad, a lo largo de todo el proyecto se ha utilizado el mismo modo de funcionamiento del SPI, adaptando el de los microcontroladores al del ADS, pues este viene impuesto por el fabricante mientras que todos los microcontroladores contemplan la utilización de varios modos alternativos.

Cabe destacar del código 5.12 que el modo de los pines denominado ESPECIAL es característico de aquellos destinados a la transmisión de información. Aunque el pin MISO se considere de entrada, si se configura en modo INPUT la comunicación fallará.

Para transmitir información a través de SPI en cualquier dispositivo de la familia de Arduino (Uno, Due, mega, etc.) se hace uso de la función SPI.transfer() o SPI.transfer16() para datos de 8 o 16 bits respectivamente.

Una de las ventajas de Arduino consiste en la estandarización en la medida de lo posible de aquellas funciones que tengan el mismo objetivo y la librería del ESP no es una excepción. Tanto la función como los ejemplos que la acompañan en la referencia oficial son compatibles con el ESP y ayudan a comprender su funcionamiento.

La transmisión por SPI desde el STM hacia el ESP se realiza en paquetes de un **byte**. Para recibir variables tipo **int** no hay ningún problema, pues al tener ese mismo tamaño se pueden transmitir directamente. Otras variables de mayor tamaño necesitan ser **divididas** en fragmentos más pequeños para ser transmitidas.

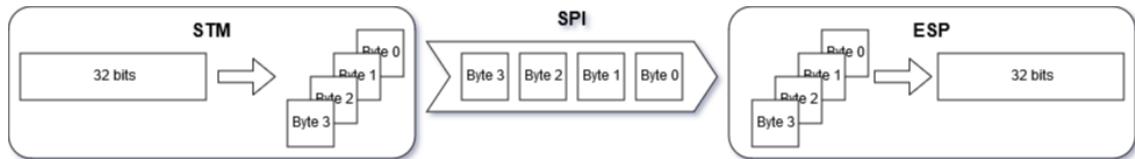


Figura 5.24: Proceso de división y ensamblado de float para su transmisión

Tras recibir los cuatro paquetes de un **byte** será necesario **ensamblarlos** para volver obtener la variable original. La misma estructura de datos utilizada para separar los datos en variables independientes se puede usar para revertir el proceso.

```

0 union miDatos{
    struct
    {
        byte b[4];      // Array de bytes de tamaño igual al tamaño de la
        // primera variable: int = 2 bytes, float = 4 bytes
    } split;
    5 float fval;
} F_recibido;
  
```

Código 5.13: Variables y tipos de dato específicos

El código 5.13 es idéntico al usado en el STM para realizar el proceso inverso ya que ambos lenguajes de programación tienen sus bases en **C**.

Esta operación se repetirá por cada una de las medidas que se realicen.

Configuración WiFi

El ESP12-E está preparado para trabajar con redes WiFi de 2.4GHz. Para poder funcionar correctamente es necesario configurar aquellos parámetros de red típicos de cualquier conexión WiFi. Estos se ven reflejados en el código 5.14.

```

0 char ssid [] = "SSID_NAME";
char pass [] = "PASSWORD";

IPAddress local_IP(192,168,4,22);
IPAddress gateway(192,168,4,9);
5 IPAddress subnet(255,255,255,0);
IPAddress remote_IP(192,168,4,121);
```

Código 5.14: Variables y tipos de dato específicos

Es importante que la contraseña de red contenga mínimo 8 caracteres o el sistema no iniciará correctamente.

Una vez configurados los parámetros básicos de comunicación es necesario iniciar la conexión. Para ello es posible iniciar dos modos distintos: conexión a red ya existente o modo *hotspot*.

La conexión a una red ya existente se realiza con el siguiente código durante la fase de configuración del ESP:

```

0 // We start by connecting to a WiFi network
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, pass)

5 while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("");

10 Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
```

Código 5.15: Conexión del ESP a una red WiFi ya existente

Como se puede apreciar, hasta que la conexión no sea satisfactoria el ESP no arrancará. Tras unos segundos el ESP acaba mostrando por pantalla la dirección IP con la que se ha conectado a la red, IP que debería coincidir con la introducida previamente en el parámetro “local_IP”.

Este modo funciona muy bien, permite conexiones rápidas y el consumo de energía es relativamente bajo, pero se hace **imprescindible la presencia de una red WiFi ya existente** cuyos parámetros sean conocidos.

El modo *Hotspot* es la otra alternativa a este modo. Con esta configuración el ESP

CAPÍTULO 5. IMPLEMENTACIÓN DEL SOFTWARE

hace las funciones de router y punto de acceso, creando una red WiFi de las características deseadas. El código para esto es el mostrado a continuación:

```
0 Serial.print("Setting soft-AP configuration ... ");
Serial.println(WiFi.softAPConfig(local_IP, gateway, subnet) ? "Ready" : "Failed!");

Serial.print("Setting soft-AP ... ");
Serial.println(WiFi.softAP("ESPBCI_WIFI", "Password_01", false) ? "Ready" : "Failed!");
5 Serial.print("Soft-AP IP address = ");
Serial.println(WiFi.softAPIP());
```

Código 5.16: Creacion de un punto de acceso en el ESP

Este modo dota al ESP de una gran **libertad e independencia** al eliminar la necesidad de una red WiFi externa, pero la gestión de la información y el consumo de energía se ven perjudicados debido al aumento de carga que recibe el microcontrolador.

Una vez establecida la conexión el ESP se queda en modo de espera de forma indefinida hasta que un nuevo cliente se conecta:

```
0 //##### Server handler #####
// Check if a client has connected
WiFiClient client = server.available();
if (!client) {
    //No client available
    5 return;
}
```

Código 5.17: Gestión de clientes y peticiones en Arduino

Tras conectarse, el cliente deberá transmitir algún tipo de información. Para evitar el cuelgue del sistema si por algún motivo el cliente no manda datos o la conexión se interrumpe, el ESP espera un tiempo prudencial y si no se ha recibido nada desecha al cliente.

```
0 // Wait until the client sends some data
Serial.println("new client");
t1 = millis();
while(!client.available()){
    delay(1);
    5 Serial.print(".");
    ESP.wdtFeed(); // Evita reinicios por watchdog
    if (millis()-t1 >= 10000){
        Serial.println("Timeout");
        return;
    10 }
}
```

Código 5.18: Gestión de clientes y peticiones en Arduino

Por último se interpreta la información almacenándola en una variable para su posterior utilización.

```
0 // Read the first line of the request  
String req = client.readStringUntil('\r\n');
```

Código 5.19: Gestión de clientes y peticiones en Arduino

Todos los microcontroladores tienen un sistema de control llamado **Watchdog** que monitoriza el estado del microcontrolador. Si detecta que el dispositivo lleva demasiado tiempo en un bucle este se activa y realiza un reinicio de emergencia.

El ESP cuenta con dos tipos distintos de Watchdog, uno accionado por software y otro por hardware. El primero se activa en caso de que el programa funcione correctamente y se haya producido un bucle no deseado o no contemplado. El segundo lo hace en caso de fallo total del sistema. Para evitar que en ciertas partes del código en las que el ESP se queda esperando de forma indefinida se active el watchdog por software es necesario introducir la siguiente línea:

```
0 ESP.wdtFeed(); // Evita reinicios por watchdog
```

Código 5.20: Reinicios por *watchdog*

De esta forma el contador que utiliza para monitorizar el tiempo que lleva en el bucle inicializa y se evita un reinicio indeseado.

Transmisión de datos a través de WiFi

Una vez se ha establecido la conexión entre ambos dispositivos se debe realizar la transferencia de información. Arduino cuenta con varias funciones asociadas a la gestión de la información tanto a nivel servidor como a nivel cliente, todas ellas englobadas dentro de la librería “WIFI”.

Si se desea usar el protocolo TCP la utilización de `client.print` resulta altamente tentadora. Esta estructura presenta ciertos **paralelismos con Serial.print** y permite además la interpretación del valor directamente al realizar una representación en formato comprensible para los humanos.

Sin embargo, al utilizar esta función se realiza también una conversión de la variable a otro formato junto con su correspondiente **pérdida de precisión y cambio en el tamaño**.

La siguiente tabla ilustra con ejemplos los resultados de estas transformaciones si se representan con 5 dígitos de precisión.

Valor real	Representación	Tamaño [bits]
1	1	8
1.3	1.33333	56 ¹

Tabla 5.3: Representación de datos usando client.print()

Al tratar con variables en las que los decimales contienen una gran cantidad de información, el uso de la función client.print() queda completamente descartado, pues provoca una pérdida de precisión muy alta o un **aumento drástico del espacio ocupado en memoria** por cada medida.

Una alternativa a esta función es client.write(). Esta es capaz de transmitir 8 bits o un fichero de datos del cliente al servidor, independientemente de la representación de esos datos.

Este sistema es menos intuitivo pero a cambio evita pérdida de precisión y asegura que **los datos transmitidos conservarán su integridad**, tanto de valor como de tipo de variable y tamaño.

La transmisión a través de TCP está orientada a asegurar que los datos llegarán sin errores a costa de disminuir la velocidad efectiva.

En un intento de mejorar la velocidad de transmisión se ha explorado la utilización de UDP usando las funciones udp.beginPacket(), udp.write() y udp.endPacket().

Los resultados se presentan en el capítulo 6, pero su inclusión al sistema ha sido desechada porque el ahorro de tiempo no es muy destacable y en cambio, en función de las condiciones de la red, la cantidad de paquetes perdidos puede resultar demasiado elevada.

¹(n*enteros + 1 (punto) + k*decimales)

5.3. LabView

El sistema en este punto puede hacer uso de cualquier ordenador o *smartphone* como interfaz. Por comodidad, se ha escogido LabView como entorno de prototipado para la interfaz, ya que incluye un gran número de funciones y elementos gráficos que simplifican el desarrollo de un proyecto y permite generar un ejecutable que funcione en cualquier ordenador, tenga o no instalada una versión de LabView.



Figura 5.25: Logotipo de LabView

5.3.1. Interfaz gráfica

Desde LabView se han habilitado funciones para la representación de la información recibida desde el sistema de adquisición de datos. En una versión inicial contiene aquellos elementos necesarios para comprobar que todo funciona correctamente y representar las señales recibidas.

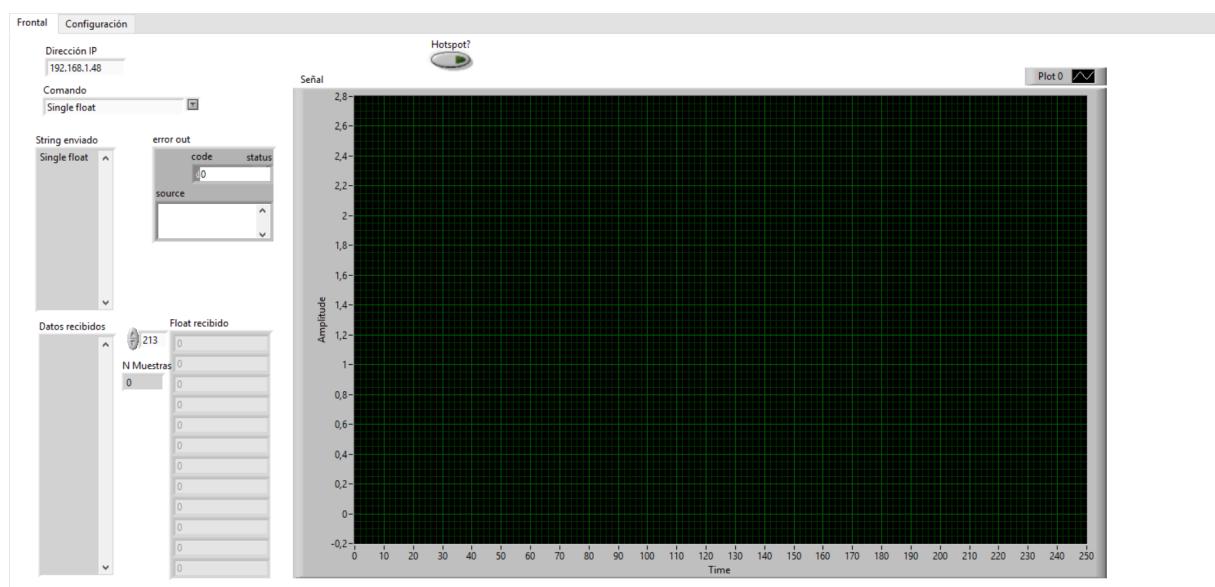


Figura 5.26: Interfaz de LabView: Representación de la información

CAPÍTULO 5. IMPLEMENTACIÓN DEL SOFTWARE

Como se puede apreciar en la imagen 5.26, la interfaz está organizada en pestañas. La primera presenta los datos recibidos, los enviados y otros elementos de control. La segunda contiene todos los elementos necesarios para modificar los parámetros de configuración del ADS.

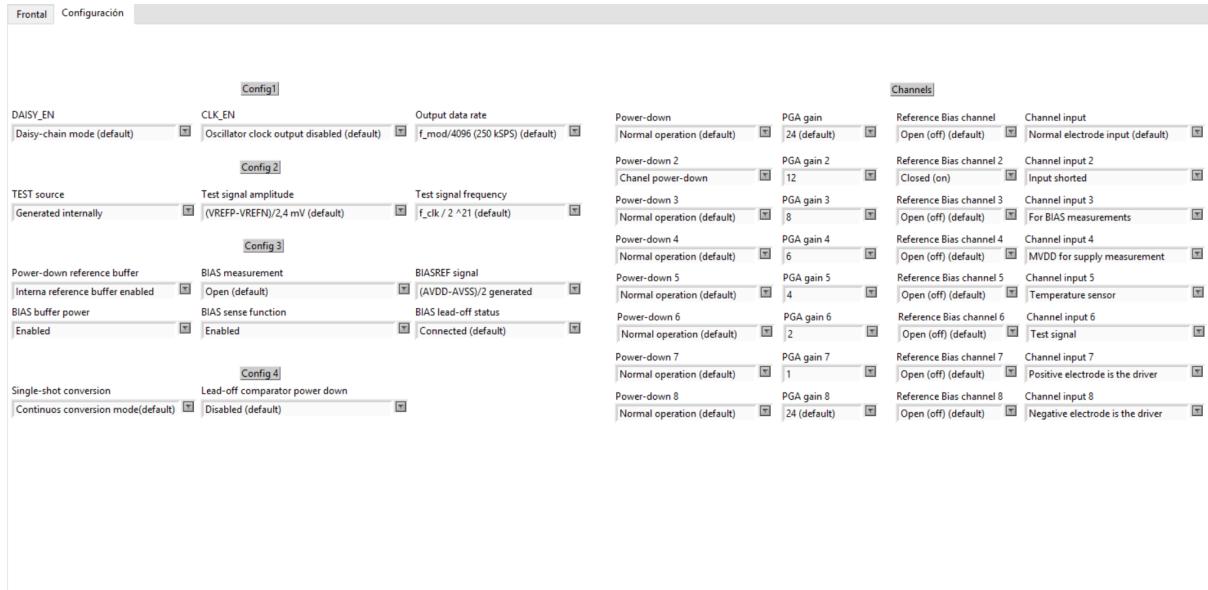


Figura 5.27: Interfaz de LabView: Configuración del ADS

La figura 5.27 muestra todos los parámetros del ADS, estando dispuesta la configuración de los canales de forma que se puedan ver todas las opciones disponibles.

Esta interfaz ha sido **desarrollada por parte de la alumna Nerea Urrestarazu** durante la realización de su Trabajo fin de Máster y cedida con el objetivo de ahorrar tiempo. El archivo .vi ha sido utilizado como base, conservando la distribución y elementos básicos relacionados con los registros y funciones del ADS mientras que la **lógica ha sido rehecha** para cumplir con las necesidades de este proyecto.

La lógica consta de varias partes, por un lado una encargada de transmitir y recibir información desde la red WiFi a la que esté conectado el ordenador. Por otro lado, otra cuya función es construir la orden que se desea transmitir a la placa adquisidora. Finalmente, tras recibir la contestación de la placa, esta se interpreta realizando las acciones pertinentes en función del significado, normalmente almacenado y representación de las mediciones.

5.3.2. Comunicación WiFi

El envío y recepción de los datos en el PC ha sido implementado de dos maneras: por TCP y por UDP.

La figura 5.28 muestra una representación simplificada de los bloques necesarios para la transmisión por TCP. Con esta configuración el ordenador inicia la conexión, manda una petición al ESP indicando que se desean medir datos y espera hasta que se reciba un paquete que contenga la cadena de caracteres retorno de carro seguido de nueva linea (\r\n).

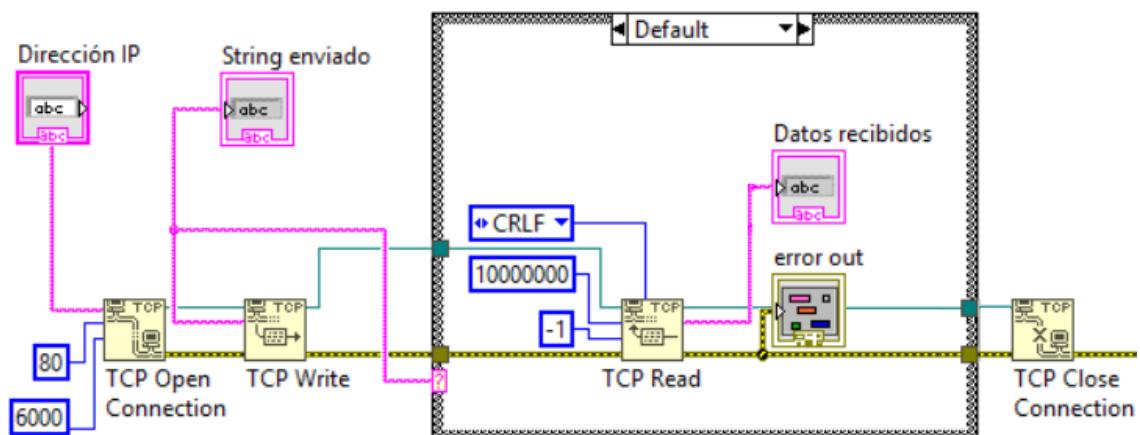


Figura 5.28: LabView: Transmisión y recepción de datos a través de TCP

En caso de que la comunicación se cierre por parte del ESP o por un error de red se genera un evento de error que será gestionado por el siguiente conjunto de bloques y mostrado al usuario sin interrumpir el resto del programa.

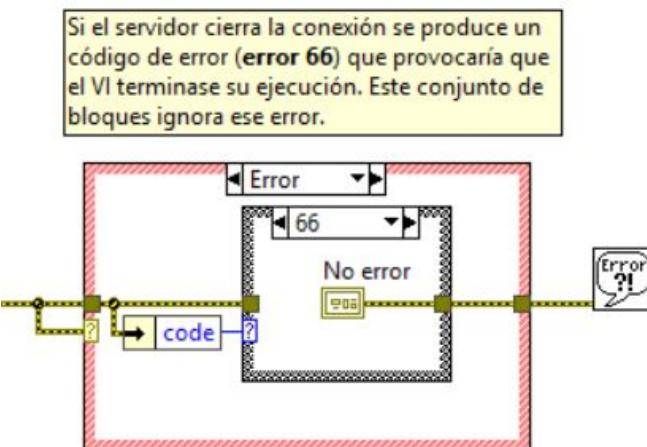


Figura 5.29: Bloques para ignorar errores conocidos

5.3.3. Conversión de datos

Los datos se reciben en un **string** de longitud igual a cuatro veces el total de los datos medidos y si estos son representados directamente resultan ilegibles. Para poder obtener los datos originales es imprescindible realizar una división cada cuatro bytes y **reinterpretar** los valores como **float** con el bloque *cast*. En la figura 5.30 se muestran los bloques encargados de realizar esta función.

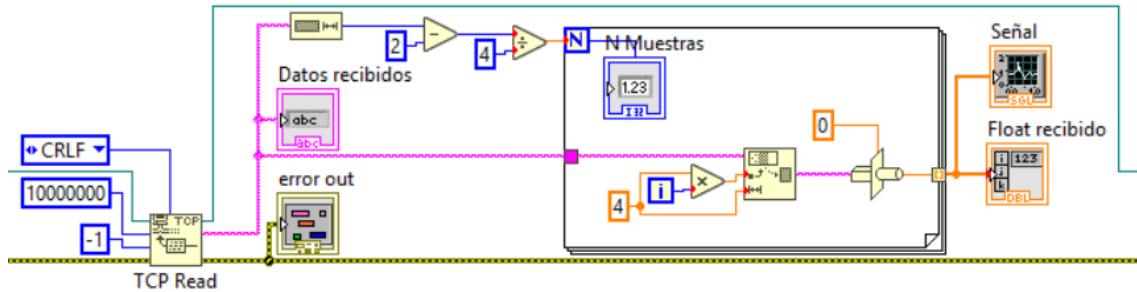


Figura 5.30: Extracción y conversión de datos

Estos bloques están pensados para funcionar independientemente del número de muestras adquiridas, aportando mayor flexibilidad al diseño.

5.3.4. Alternativa UDP

Como se ha mencionado anteriormente, se ha trabajado en una alternativa que aprovecha las ventajas del UDP para transmitir la información hacia el ordenador. Los bloques utilizados para realizar esta función los siguientes:

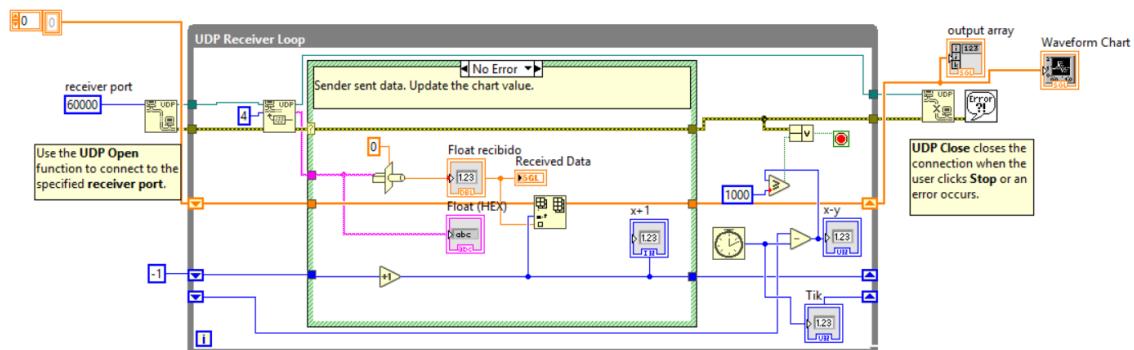


Figura 5.31: Recepción de datos a través de UDP

Este sistema funciona de forma similar al equivalente en TCP. En esta ocasión que en lugar de esperar a un carácter concreto para dar la comunicación por concluida esta se cierra tras un tiempo de espera sin recepción de datos. De esta forma se evita un bloqueo en caso de que se pierda alguna muestra.

6

Resultados

En el presente capítulo se han recopilado los resultados obtenidos tras cada una de las fases del proyecto. Por motivos de comprensión y presentación, **aquellos resultados asociados al desarrollo de la PCB han sido presentados en el capítulo 4** de modo que no se volverán a repetir en este. En caso de que se deseen revisar de nuevo, estos se encuentran en el apartado 4.5.

Los siguientes apartados contienen aquellos resultados asociados al desarrollo del software tanto del microcontrolador STM32F4 como del ESP y su comunicación con LabView.

6.1. Funcionamiento del ADS

Para comprobar que el ADS está midiendo correctamente este incluye una señal de *test* conocida cuyas características se encuentran perfectamente definidas en la hoja de características del componente.

Las características de la señal *test* dependen de los bits almacenados en el registro **Config 2**. Durante la realización de este proyecto se le ha asignado a Config 2 el valor 0xD1 (11010001) para que la señal de *test* se genere de forma interna, con un voltaje que oscila entre $\pm 4,5/2400V$ ($\pm 1,875mV$) y frecuencia de f_{CLK}^{20} .

La siguiente figura muestra una representación del voltaje real de la señal *test* comparándolo con la amplitud teórica calculada según la fórmula proporcionada por la hoja de características:

CAPÍTULO 6. RESULTADOS

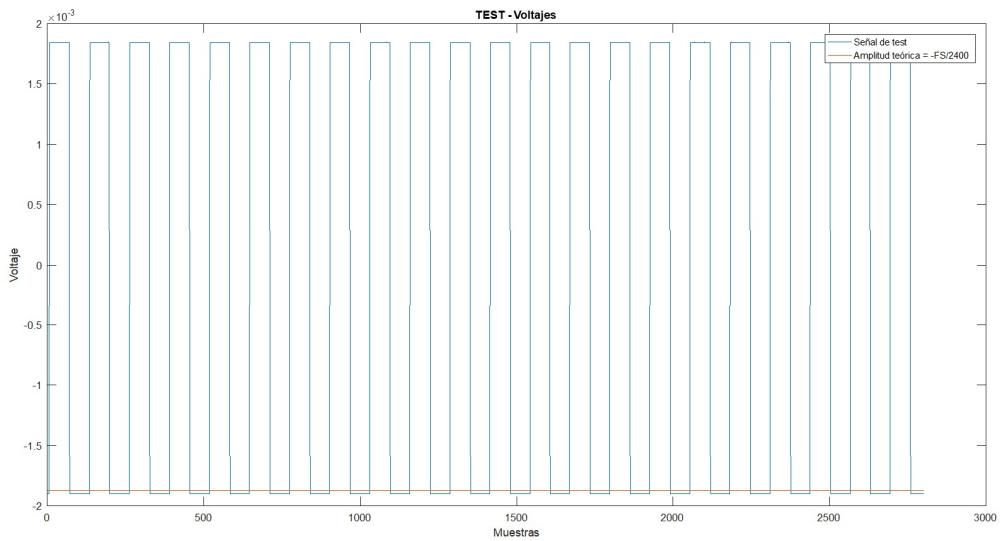


Figura 6.1: Comparativa entre la señal test y su amplitud teórica

Finalmente, tras realizar una medida desde Labview el resultado es el siguiente:

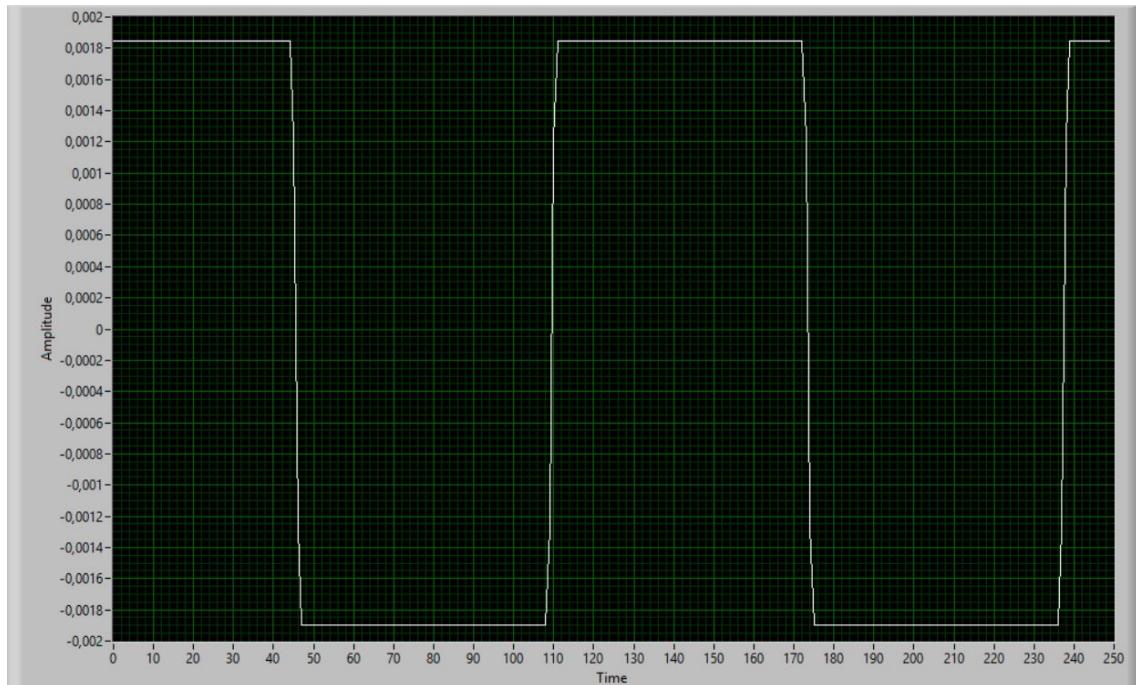


Figura 6.2: Señal de test medida usando LabView

Tras comprobar que se recibe una señal con las características esperadas se puede concluir que el ADS se encuentra correctamente configurado y listo para realizar medidas sobre voltajes reales.

6.2. Medidas sobre señales reales

Las primeras señales reales a medir han sido señales con características conocidas. De esta forma resulta muy sencillo identificar si algún elemento no está funcionando correctamente. Para una prueba inicial se preparó un generador de señales haciendo uso de un Arduino UNO y se generó una señal de amplitud entre 0V y 5V y frecuencia de 2Hz. El resultado tras medir esta señal se encuentra representado en la figura 6.3.

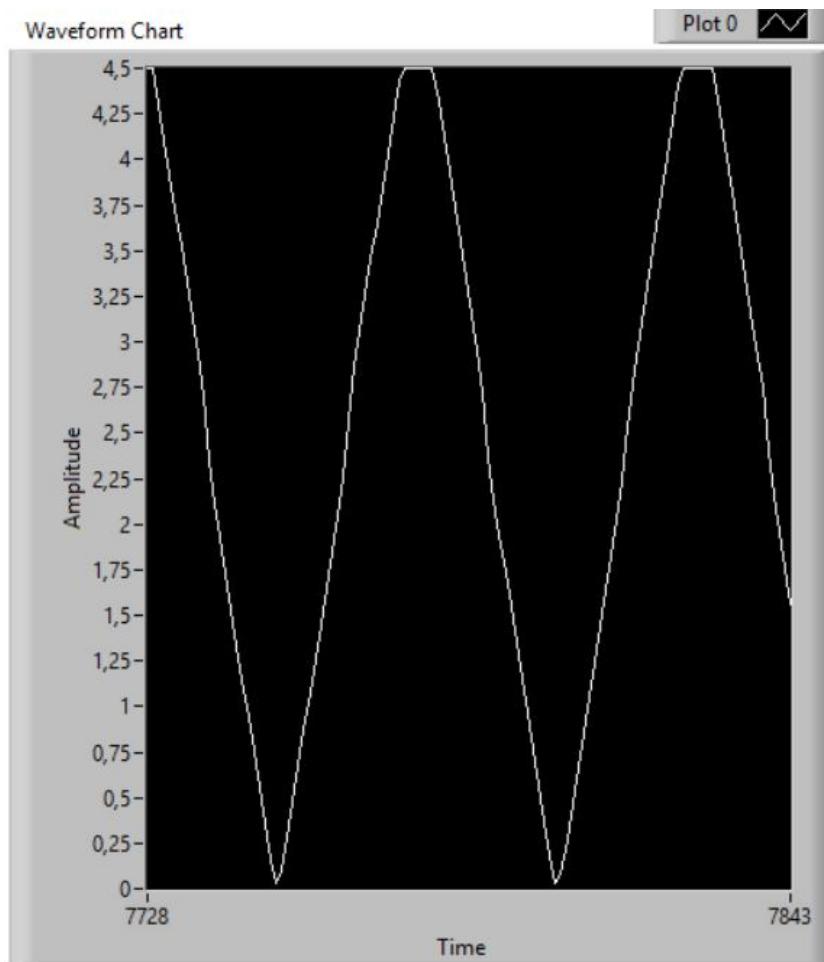


Figura 6.3: Señal real triangular generada con Arduino

La señal se está adquiriendo correctamente salvo por una pequeña deformación en los picos superiores del triángulo. Este efecto es debido a la saturación del sistema, pues la amplitud máxima que es capaz de adquirir tanto en positivo como en negativo depende directamente de V_{REF} (4.5V).

6.3. Filtrado de la señal

El siguiente paso es aplicar a esas señales reales un filtrado para comprobar que los filtros implementados cumplen con su función.

Siguiendo el ejemplo sugerido por la referencia de CMSIS-DSP se implementó un filtrado de rechazo de banda con el que eliminar las componentes en frecuencia no deseadas de una señal. La señal original es una señal artificial formada por dos senos de distintas frecuencias (10Hz y 50Hz) y el objetivo es eliminar la componente de 50Hz o minimizar su impacto.

En primer lugar se realizó una estimación del efecto que tendría aplicar un filtro sobre una señal de esas características. El resultado de la simulación realizada en MatLab es el mostrado en la figura 6.4

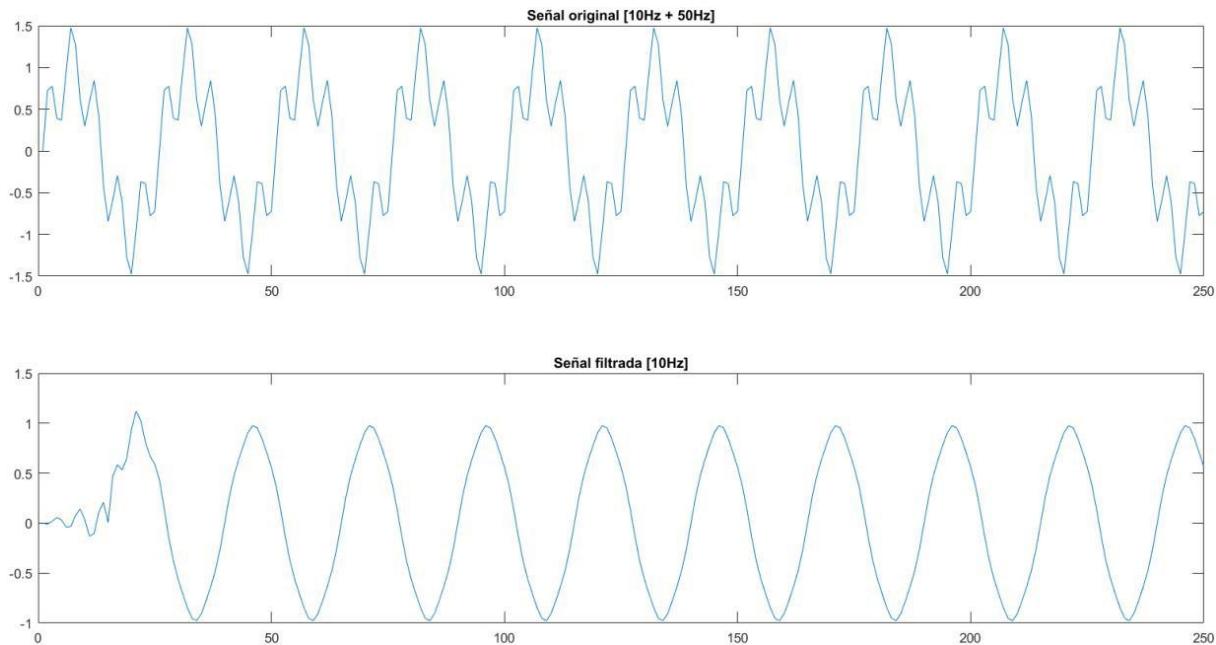
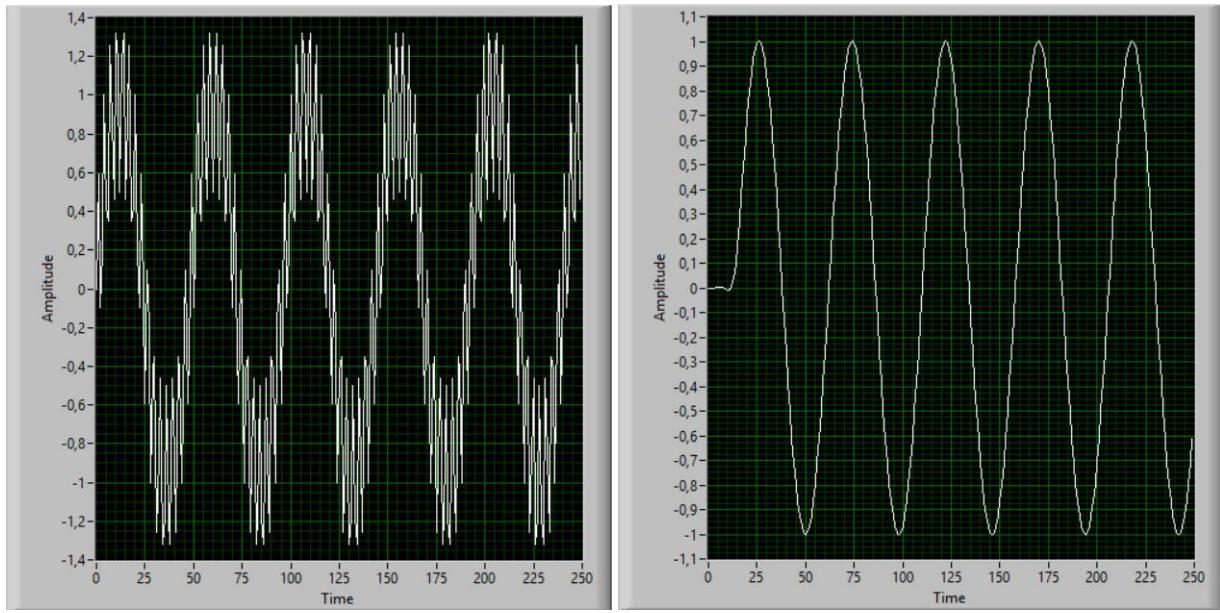


Figura 6.4: Simulación del efecto del filtro Notch a 50Hz

Como se puede observar, la componente de 50Hz **ha sido eliminada** casi por completo a costa de afectar ligeramente a la amplitud de la señal original. Tras comprobar que el resultado es satisfactorio se procedió a realizar una implementación del mismo filtro haciendo uso de CMSIS-DSP y el microcontrolador STM32F4.

Al no contarse con generadores de señales capaces de producir una señal de estas características se optó por generar una señal artificial con MatLab y almacenarla en la memoria del STM en la fase de programación. La figura 6.5 muestra la señal original (a) y la señal tras aplicarle el filtro (b).



(a) Señal original con ruido de 50Hz

(b) Señal filtrada

Figura 6.5: Implementación real del efecto del filtro Notch a 50Hz

Tal y como se esperaba, la señal de 50Hz ha sido eliminada completamente, siendo los **resultados** obtenidos prácticamente **idénticos a los simulados**.

El diseño del filtro fuerza a que las componentes en frecuencia cercanas a 50Hz se vean más atenuadas que aquellas más alejadas. Las siguientes figuras muestran señales de 5Hz, 10Hz, 50Hz y 65Hz y como el cambio de frecuencia afecta a la amplitud de la señal.

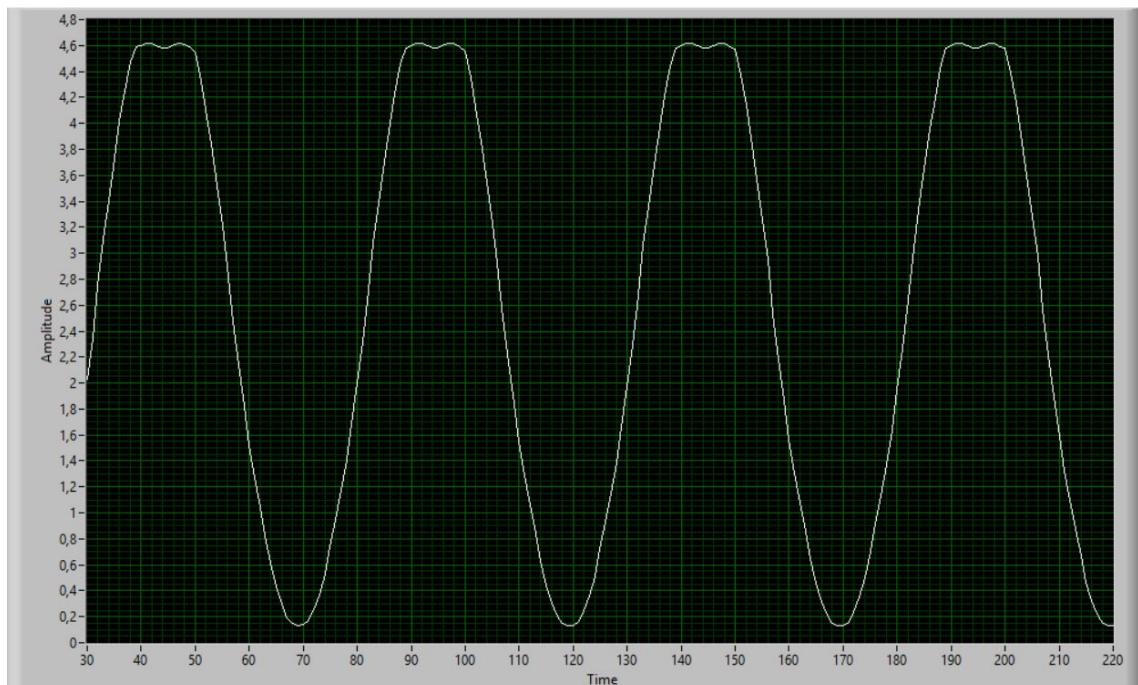


Figura 6.6: Señal de 5Hz filtrada

CAPÍTULO 6. RESULTADOS

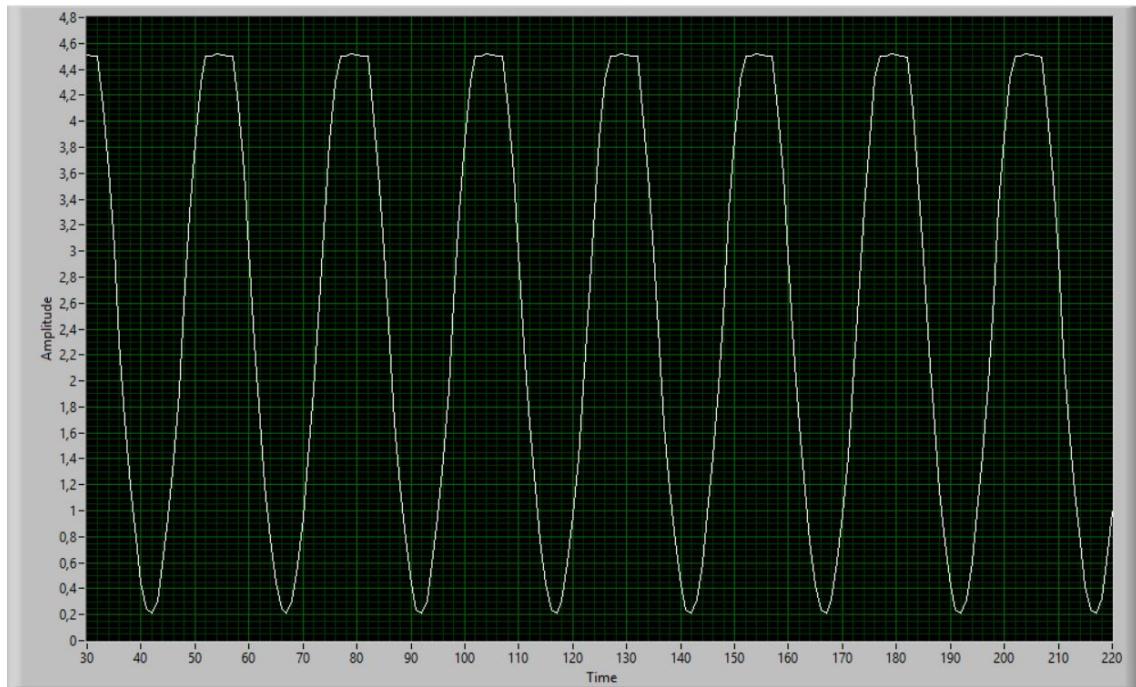


Figura 6.7: Señal de 10Hz filtrada

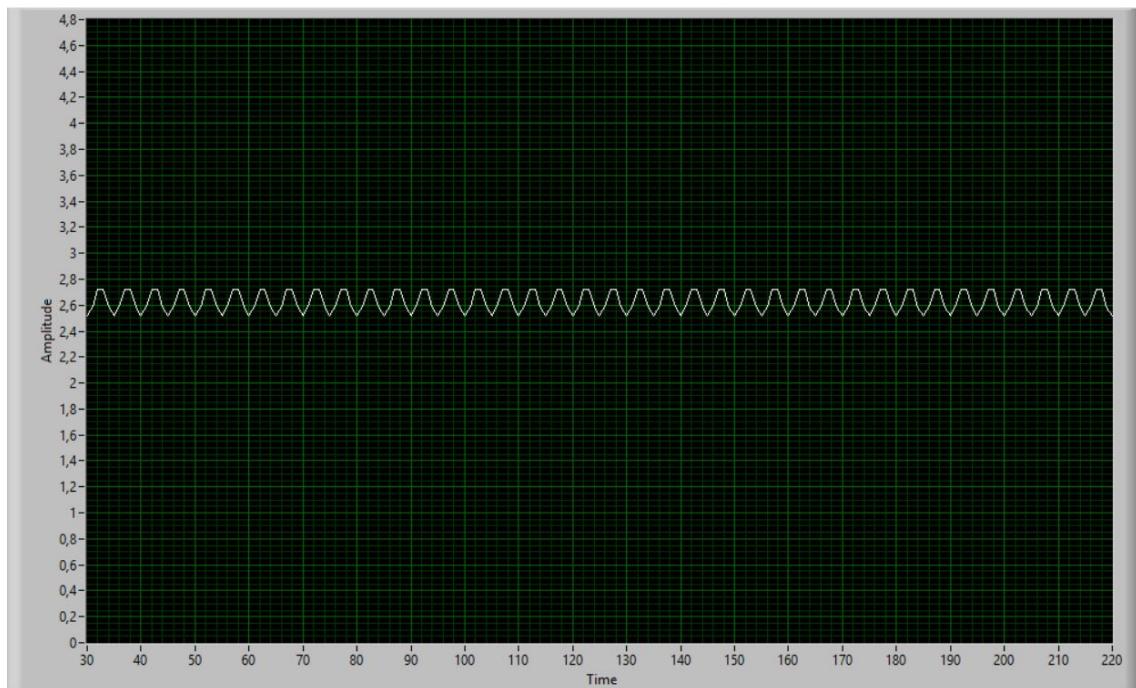


Figura 6.8: Señal de 50Hz filtrada

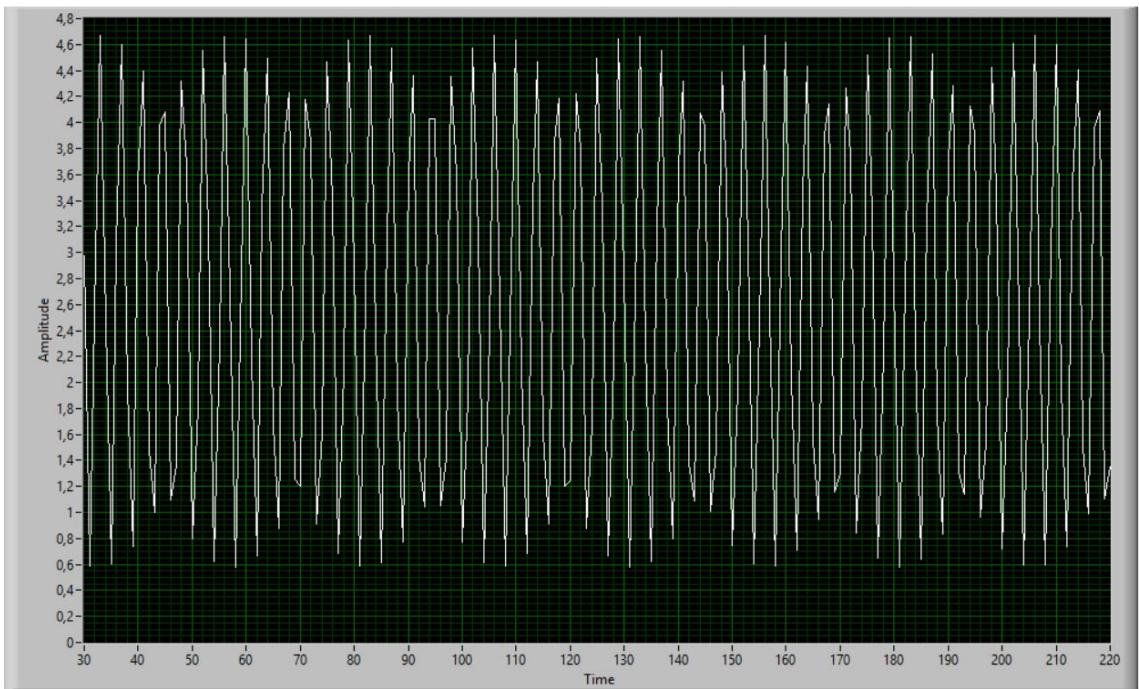


Figura 6.9: Señal de 65Hz filtrada

El sistema está cumpliendo su función, pues ha atenuando claramente la señal de 50Hz mientras que las demás se mantienen con una amplitud similar a la original.

6.4. Medida de un EEG real

Las siguientes figuras representan un EEG medido usando la placa de adquisición tras el filtrado de la componente de 50Hz.

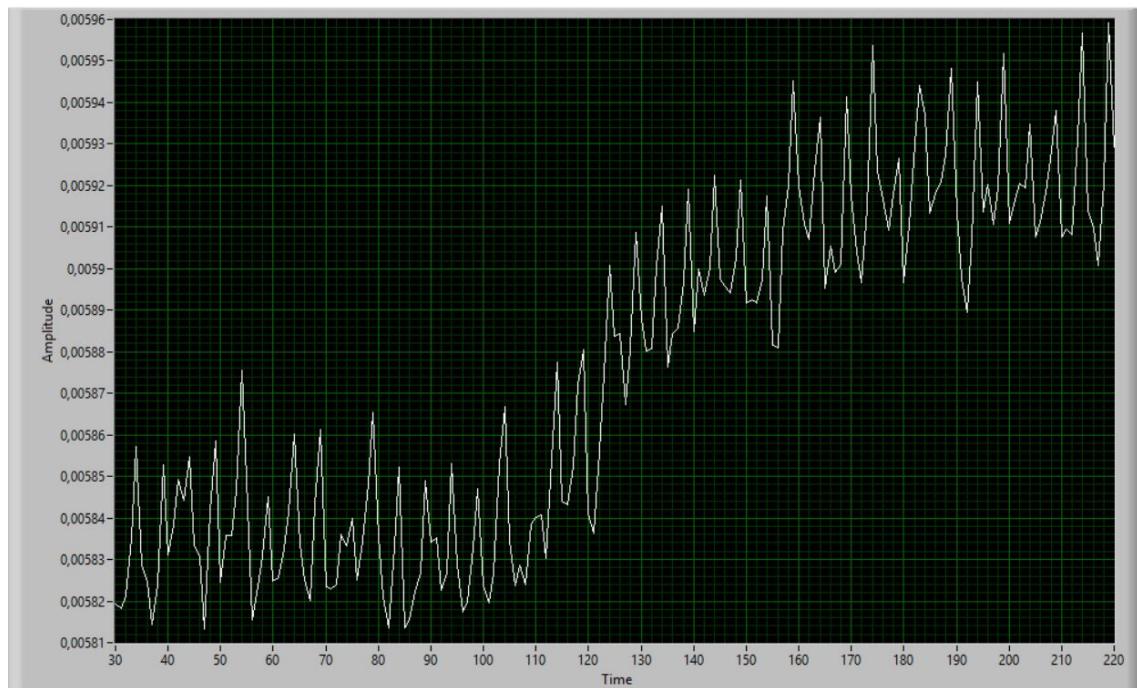


Figura 6.10: EEG abriendo los ojos

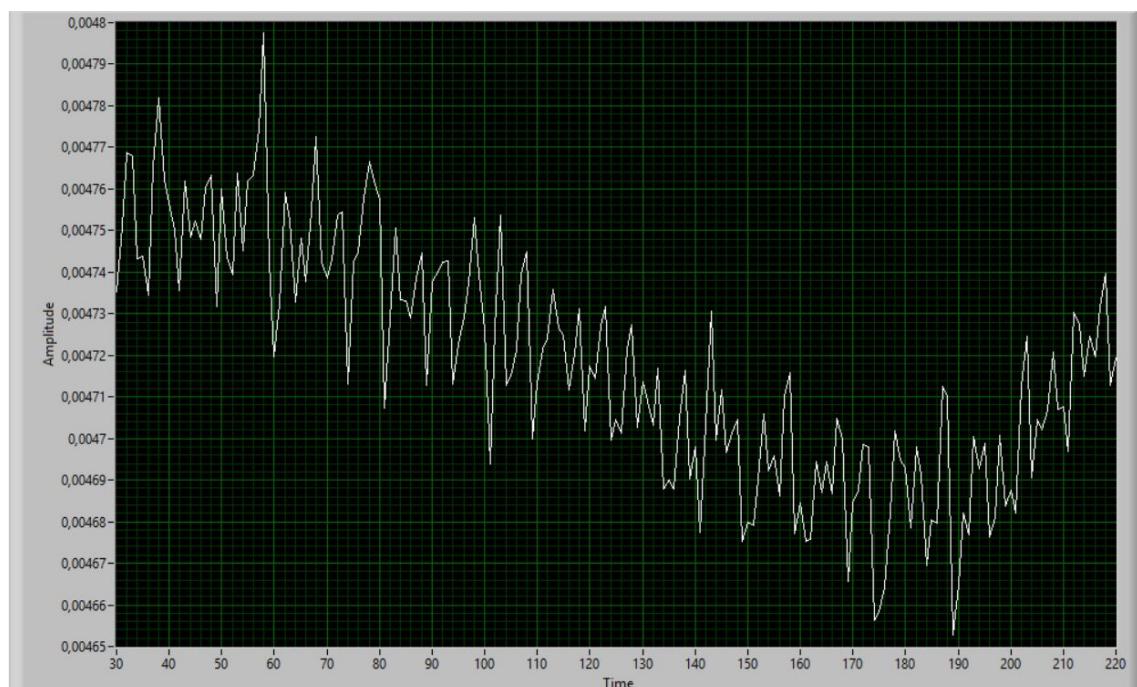


Figura 6.11: EEG con los ojos cerrados

A pesar del filtrado que se ha implementado aún se sigue observando una fuerte componente en frecuencias cercanas a los 50Hz. Esto se debe en gran parte a los sistemas de medida utilizados, ya que la presencia de cables tan largos, finos y con tan poco blindado los convierte en antenas perfectas para captar ruido de esa frecuencia.

La figura 6.12 muestra el sistema de medida utilizado para adquirir estas señales.



Figura 6.12: Sistema de medida utilizado para la adquisición del EEG

6.5. Transmisión de datos por WiFi

Por último se incluirán los resultados obtenidos mediante la transmisión de información por TCP y UDP a través de WiFi.

La librería WiFi del ESP contienen un gran número de funciones relacionadas con la transmisión de información tanto por UDP como por TCP. Con el objetivo de transmitir los datos de la forma más **fiable** posible se optó por transmitirlos a través de **TCP**, pues este protocolo tiene sistemas que aseguran que el paquete llegará al destino.

Sin embargo utilizar este protocolo se tradujo en un tiempo de transmisión por cada 250 muestras superior a los 2 minutos haciendo inviable su uso.

En vista de la eficiencia de transmisión conseguida con TCP se utilizó **UDP** para la transmisión de la información al ordenador. Haciendo uso de este sistema la transmisión se completaba en apenas 1 segundo pero no todos los paquetes llegaban. Tras un poco de investigación se observó que cuanto mayor fuese el retardo introducido entre paquetes menos paquetes se perdían así que se afinó la transmisión hasta conseguir una tasa de **transferencia lo suficientemente rápida con una pérdida de paquetes mínima**.

CAPÍTULO 6. RESULTADOS

En las siguientes imágenes se muestran tres casos en los que se transmitieron 100 paquetes que contenían variables de tipo float, el retardo utilizado entre paquetes y el porcentaje de paquetes perdidos.

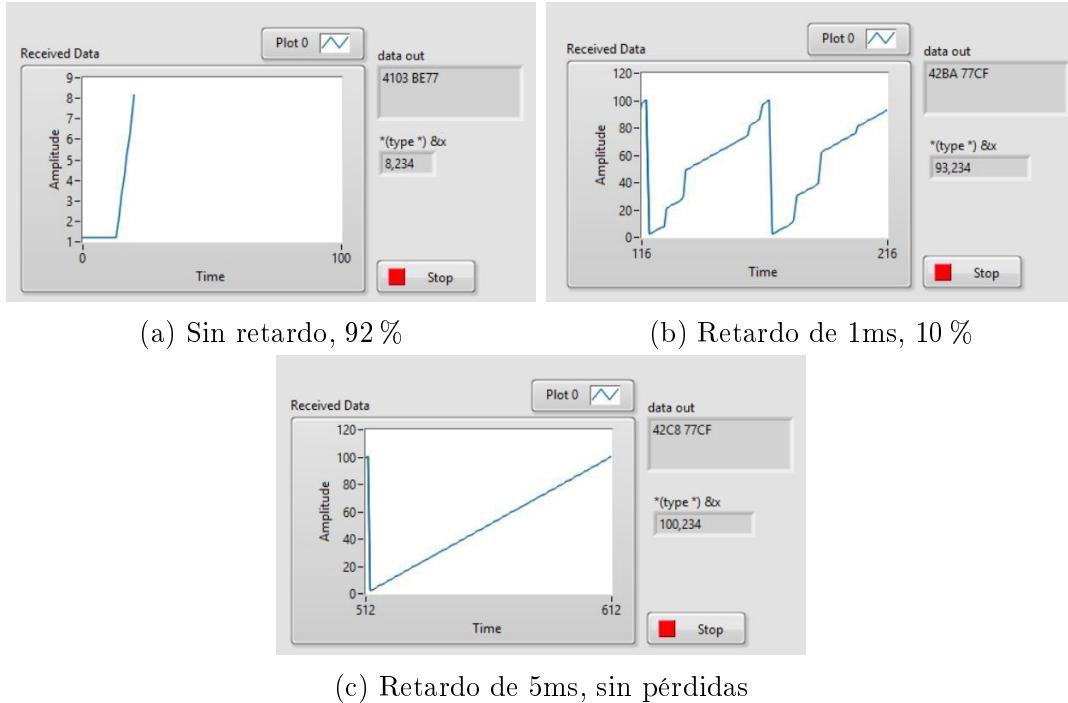


Figura 6.13: Comparativa entre los distintos retardos y el porcentaje de paquetes perdidos

Con esta información se deduce que para que se reciban todas las muestras que se adquieren por segundo el tiempo de transmisión asciende a 1.25s (250 muestras * 0.005 segundos/muestra).

Este sistema sería ideal si entre transmisiones el sistema de adquisición capturase la siguiente muestra, pues permitiría montar un sistema de adquisición continuo. Por desgracia esto no es compatible con el filtrado de la señal, al menos no directamente. Además la posibilidad de perder muestras es una característica no deseable.

Finalmente se realizó una segunda iteración sobre TCP, revisando en profundidad las librerías se detectó un error que provocaba un aumento en el tiempo de transmisión muy elevado (es importante recordar que **dichas librerías no son oficiales** y están en constante desarrollo por lo que la presencia de errores menores es algo habitual). Tras solventar dicho error el tiempo de transmisión de 250 muestras bajó a 1.28 segundos.

El resultado final es muy cercano al conseguido con UDP sin la posibilidad de perder muestras como ocurría con ese protocolo.

7

Conclusiones

Para la realización de este proyecto ha sido necesario aplicar conocimientos relacionados con casi todas las ramas de las telecomunicaciones.

La rama **electrónica** es claramente la **dominante** durante la realización del proyecto, pues el diseño del esquemático, la PCB, y el montaje de todo el sistema sería imposible sin los conocimientos adquiridos durante mi estancia en la Universidad. Aun así, es posible apreciar ciertos **matices relacionados con el resto de las ramas**. Gestión de redes, protocolos de comunicación y el filtrado de las señales han sido utilizados para conseguir los resultados ya mostrados.

El diseño de la PCB fue muy interesante ya que hasta este momento todos los diseños realizados habían sido orientados al análisis teórico y simulaciones, dejando de lado la **implementación final de un sistema** de esas características.

Una de las partes del proyecto que **más trabajo** ha supuesto y que menos se ve reflejado en esta memoria o en el resultado final es el **desarrollo del firmware del microcontrolador STM32F4**. Este microcontrolador tiene una gran comunidad que lo respalda y un buen número de referencias, pero carece de un sistema de aprendizaje autónomo o guías de iniciación. Esto provoca que el **comienzo** del desarrollo resulte especialmente **complicado**.

El sistema ha cumplido ampliamente con las expectativas. Presenta además **mejores características y un presupuesto muy inferior** al de su competidor directo, OpenBCI. El coste final del proyecto es de 113€¹ contando la tarjeta de adquisición y trabajando con 16 canales, mientras que si se desea utilizar OpenBCI con WiFi y es necesario hacer un desembolso de 300€ y sólo puede trabajar con 4 canales, 8 en el mejor de los casos.

¹El presupuesto se explica con más detalle en el apéndice A

7.1. Futuras mejoras

Por supuesto, aunque el proyecto final ha sido un éxito, no se debe olvidar que este sistema es un **prototipo inicial** y que no se había trabajado antes con ciertos elementos que juegan un papel principal de modo que es natural que presente ciertos puntos que podrían ser mejorables en el futuro.

En futuras iteraciones sobre este proyecto sería deseable **ampliar las características** ya presentes dotando al sistema de ciertas funciones que no se han implementado por falta de tiempo y recursos.

Algunas de estas características son la adquisición en tiempo real, la parametrización del número de muestras y los tipos de filtros y la inclusión en el software de un sistema de almacenamiento integrado (ya contemplado en el hardware). Igualmente, añadir el **sistema de debug por JTAG** facilitaría el desarrollo del firmware del microcontrolador.

La mejora más importante podría ser el desarrollo de una PCB en la que se encuentren **todos los elementos integrados**, aumentando la comodidad al trabajar con el sistema y su portabilidad.

Glosario

Full Duplex Cualidad de los elementos que permiten la entrada y salida de datos de forma simultánea. 26, 63

Open Source Filosofía aplicada a la realización de un proyecto (Software o Hardware) que implica una colaboración abierta, proporcionando aquella documentación necesaria para replicar dicho proyecto y proporcionando la libertad de aprovechar y/o mejorarlo sin restricciones. 3

Tensión de Dropout Mínima diferencia de tensión entre la entrada y la salida dentro de la cual el circuito es todavía capaz de regular la salida dentro de las especificaciones. 28

Acrónimos

ADC Convertidores Analógico-Digital. 20, 23, 24, 29, 30

CPU central processing unit. 22, 24

CSS Clock Security System. 55

DSP Digital Signal Processing. 25, 26

ECG Electrocardiograma. 12

EEG Electroencefalograma. 4, 6, 8, 12–15, 19, 20, 113

FCPU Frecuencia de la CPU. 26

FIR Finite Impulse Response. 72

FPU Floating Point Unit. 26

GPIO General Purpose Input/Output. 20, 31, 61

HAL Hardware Abstraction Layer. 53, 59

HSE High-Speed External clock. 55

IDE Integrated Development Environment. 24, 34, 51, 57, 59, 76

jtag Joint Test Action Group. 100

LED Light-Emitting Diode. 30, 31

LQFP64 64 pin Low-profile Quad Flat Package. 26

LSI Low-Speed Internal clock. 55

MCU Microcontroller Unit. 22, 26, 32, 33

OTG On The Go. 26

PCB Print Board Circuit. 3, 22, 29, 33, 35–37, 39, 40, 42, 44, 46, 89, 99, 100

PLL Phase-Locked Loop. 32, 55

RAM Random Access Memory. 22, 24, 26

SMT Surface-Mount Technology. 3, 39

SNR Signal-to-Noise Ratio. 8

SoC System on Chip. 22

SPI Serial Peripheral Interface. 20, 22, 23, 26, 29, 32, 34, 61, 62

SPS Samples Per Second. 25

THT Through-Hole Technology. 39

UART Universal Asynchronous Receiver-Transmitter. 22, 23

USB Universal Serial Bus. 21, 24, 26, 33

Bibliografía

- [1] Jose Javier Serrano Olmedo. *Apuntes de la asignatura de Señales e imágenes médicas*. 2017.
- [2] Fundación Wikimedia, Inc. *Wikipedia - La enciclopedia libre*. 2018. URL: <https://es.wikipedia.org>.
- [3] cah6. *DIY EEG (and ECG) Circuit*. Ago. de 2012. URL: <http://www.instructables.com/id/DIY-EEG-and-ECG-Circuit/>.
- [4] *Building an EEG at home with the OpenEEG project*. 2018. URL: <http://openeeg.sourceforge.net/buildeeg/>.
- [5] Io Flament. *Make an EEG Beanie That Reads Your Mind*. URL: <https://makezine.com/projects/make-43/mind-reading-beanie/>.
- [6] OpenBCI. *OpenBCI - An Open-Source Brain-Computer Interface*. URL: <https://github.com/OpenBCI>.
- [7] Texas Instrument. *ADS1299-x Low-Noise, 4-, 6-, 8-Channel, 24-Bit, Analog-to-Digital Converter for EEG and Biopotential Measurements*. 2018. URL: www.ti.com/lit/ds/symlink/ads1299.pdf.
- [8] *esp8266-wiki - Boot Process*. 2018. URL: <https://github.com/esp8266/esp8266-wiki/wiki/Boot-Process>.
- [9] 2018. URL: <http://www.instructables.com/id/GETTING-STARTED-WITH-IoT-With-ESP8266-MicroControl/>.
- [10] Simblee. *Simblee RFD77101*. 2018. URL: <https://www.mouser.com/ds/2/470/Simblee%20RFD77101%20Datasheet%20v1.0-786079.pdf>.
- [11] Texas Instrument. 2018. URL: <http://www.ti.com>.
- [12] STMicroelectronics. 2018. URL: <http://www.st.com>.
- [13] STMicroelectronics. *STM32F405xx and STM32F407xx*. 2018. URL: <http://www.st.com/content/ccc/resource/technical/document/datasheet/ef/92/76/6d/bb/c2/4f/f7/DM00037051.pdf/files/DM00037051.pdf/jcr:content/translations/en.DM00037051.pdf>.
- [14] Diodes Incorporated. *AZ1117C*. 2018. URL: <https://www.mouser.es/datasheet/2/115/AZ1117C-1147948.pdf>.

BIBLIOGRAFÍA

- [15] STM Electronics. *Oscillator design guide for STM8S, STM8A and STM32 microcontrollers*. STM Electronics. Ago. de 2015. URL: http://www.st.com/content/ccc/resource/technical/document/application_note/c6/eb/5e/11/e3/69/43/eb/CD00221665.pdf/files/CD00221665.pdf/jcr:content/translations/en.CD00221665.pdf.
- [16] *Normal Condition of PCB capabilities*. ITEAD Studio. 2018. URL: <http://support.iteadstudio.com/support/solutions/articles/1000156313-normal-condition-of-pcb-capabilities>.
- [17] *10k Ohm 0603 Resistor*. 2018. URL: Unique-Leds.com.
- [18] STMicroelectronics. *STM32 IDEs*. 2018. URL: <http://www.st.com/en/development-tools/stm32-ides.html?querycriteria=productId=LN1200>.
- [19] *STM32F4xx DSP and Standard Peripherals Library - Peripheral Drivers Description*. 2013. URL: http://dsr-company.com.amm/STM32_docs/CHM/Library_Files.html.
- [20] Arm Ltd. *CMSIS DSP Software Library*. Feb. de 2018. URL: <http://www.keil.com/pack/doc/CMSIS/DSP/html/index.html>.
- [21] F.S. Schlindwein. *FIR filter design using the Fourier series method*. Dic. de 1995.
- [22] AI-Thinker. *ESP-12E WiFi Module*. 2018. URL: <https://www.kloppenborg.net/images/blog/esp8266/esp8266-esp12e-specs.pdf>.
- [23] Microchip. *150 mA Ultra-Low Quiescent Current, Capacitorless LDO Regulator*. 2018. URL: <http://www.microchip.com/mymicrochip/filehandler.aspx?ddocname=en574465>.
- [24] Electrónica Unicrom. *Reguladores de voltaje monolíticos*. 2016. URL: <https://unicrom.com/reguladores-de-voltaje-monoliticos/>.
- [25] Kicad EDA. 2018. URL: <http://kicad-pcb.org/>.
- [26] Shawon Shahryiar. *STM32 - Prior to start*. Dic. de 2014. URL: <http://embedded-lab.com/blog/stm32-prior-to-start/>.
- [27] tilz0R. *Program STM32F4 with UART*. 2015. URL: <http://stm32f4-discovery.net/2014/09/program-stm32f4-with-uart/>.
- [28] brian. *Getting Started with the ESP8266*. Jun. de 2017. URL: <http://pratumlabs.com/blog/2017/06/getting-started-with-the-esp8266/>.
- [29] Arduino for STM32. 2018. URL: <http://stm32duino.com/>.
- [30] Carlos Novo,Leticia Chacón Gutiérrez,José Alberto Barradas Bribiesca. *Mapeo Electroencefalográfico y Neurofeedback*. Feb. de 2010. URL: https://www.researchgate.net/figure/Figura-3-Sistema-Internacional-10-20-para-la-colocacion-de-los-electrodos_fig2_282294960.
- [31] conorrussomanno. *Arduino Library for the Texas Instruments ADS1299*. URL: <https://github.com/conorrussomanno/ADS1299>.
- [32] Ace Medlock, Kendrick Shaw, Eric Herman. *OpenHardwareExG*. URL: <https://github.com/openelectronicslab/OpenHardwareExG>.

- [33] Carmine Noviello. *Mastering STM32*. Leanpub, nov. de 2016. URL: <http://leanpub.com/mastering-stm32>.
- [34] Geoffrey Brown. *Discovering the STM32 Microcontroller*. Ene. de 2013.
- [35] *Restriction of Hazardous Substances*. 2018. URL: <http://www.asiap.org/AsIAP/index.php/raee/300-articulos/3004-que-es-el-rohs-y-por-que-es-importante>.

Apéndices

A

Bill of Materials (BOM)

A continuación, en la tabla A.1 se detalla el desglose de todos los componentes necesarios para el desarrollo de este proyecto. Este documento recibe comúnmente el nombre de *Bill Of Materials* y permite realizar una estimación aproximada del coste del producto final.

Reference	Value	Footprint	manf#	Qty	Precio[€]/unidad	Precio total: (EUR)
R6, R17	220	Resistors_SMD:R_0603	RC0603JR-07220RL	2	0,094 €	0,19
C13, C17, C19, C21, C24, C6, C8, C9	100n	Capacitors_SMD:C_0603	CL10B104K08NNNC	8	0,217 €	1,74
R10, R2, R3, R4, R8	10k	Resistors_SMD:R_0603	RC0603JR-0710KL	5	0,008 €	0,04
C12, C16, C18, C20,C22	10n	Capacitors_SMD:C_0603	CGJ3E2X7R1C103K080AA	5	0,140 €	0,70
C14, C15	10p	Capacitors_SMD:C_0603	GRM0335C1H100GA01D	2	0,094 €	0,19
R1	1k	Resistors_SMD:R_0603	RC0603JR-131KL	1	0,094 €	0,09
R11	1M	Resistors_SMD:R_0603	CRCW06031M00FKEA	1	0,094 €	0,09
D1, D2	1N4148	Diodes_SMD:D_MiniMELF_Standard	LL4148	2	0,094 €	0,19
C1, C2, C23, C4, C5	1u	Capacitors_SMD:C_0603	CL10A105MQ8NNNC	5	0,072 €	0,36
C25, C26	2.2p	Capacitors_SMD:C_0603	02013A2R2BAT2A	2	0,094 €	0,19
R12	1.8k	Resistors_SMD:R_0603	CRCW06031K80FKEA	1	0,094 €	0,09
Q1	2N7002	TO_SOT_Packages_SMD:SOT-23	2N7002E	1	0,236 €	0,24
C10, C11, C3, C7	4.7u	Capacitors_SMD:CP_Elec_4x4.5	865090340001	4	0,193 €	0,77
R5, R7, R9	47k	Resistors_SMD:R_0603	RC0603JR-0747KL	3	0,008 €	0,02
F1	500mA	Resistors_SMD:R_0603	06035FV050F/32-2	1	1,030 €	1,03
U2	AZ1117CH-3.3	TO_SOT_Packages_SMD:SOT-223	AZ1117CH-3.3TRG1	1	0,246 €	0,25
CON1	BARREL_JACK	bci-base_library:BARREL_JACK	EJ08A	1	0,870 €	0,87
P1, P2	CONN_01X02	Connectors_Molex_KK-6410-02_02x2.54mm_Straight	640454-2	1	0,094 €	0,09
P3	CONN_01X05	Pin_Headers:Pin_Header_Straight_1x05_Pitch2.54mm	61300511121	3	0,230 €	0,69
P4, P5	CONN_02X04	Pin_Headers:Pin_Header_Straight_2x04_Pitch2.54mm	M20-9980446	2	0,273 €	0,55
Y1	Crystal	Crystals:Crystal_SMD_0603-2pin_6.0x3.5mm	F60800010	1	0,886 €	0,89
U4	ESP-12E	bci-base_library:ESP-12E	ESP-12E	1	2,500 €	2,50
D3	LED	LEDs:LED_0603	LTS1-C190KGKT	1	0,264 €	0,26
D4	LED	LEDs:LED_0603	SML-D12V1WT86	1	0,189 €	0,19
U1	MCP1711T-50	TO_SOT_Packages_SMD:SOT-23-5	MCP1711T-50/OT	1	0,396 €	0,40
U3	RFD77101	doragasu-footprints:RFD77101	RFD77101	1	17,110 €	17,11
U5	STM32F405x_LQFP64	Housings_QFP:LQFP-64_10x10mm_Pitch0.5mm	STM32F405RGRT6	1	9,970 €	9,97
SW1	SW_Push	open-project:SW_PUSH_SMD	2-1437565-8	1	0,189 €	0,19
P6	USB_A	Connectors:USB_A	KUSBE-ASFS1N-W	1	0,605 €	0,61
R13, R14, R15, R16	NP	Resistors_SMD:R_0603	-	4	-	-
					Total	40,49

Tabla A.1: BOM - Bill of Materials

El coste final contando exclusivamente los materiales asciende a 40,49€.

APÉNDICE A. BILL OF MATERIALS (BOM)

Es importante destacar que el coste de la tarjeta de procesado no sustituye al de la tarjeta de adquisición sino que lo complementa. Si se desea recrear este proyecto desde cero será imprescindible comprar todos los componentes y montar ambas tarjetas. Teniendo en cuenta el presupuesto proyectado para la tarjeta de adquisición (97€), el de la tarjeta de procesado (40,49€) y que ciertos componentes como los módulos Bluetooth, WiFi y USB de la tarjeta de adquisición no son necesarios, **el precio final del sistema completo es inferior a los 113€.**

B

Impacto ético, económico, social y ambiental

El objetivo principal de este proyecto consiste en construir un sistema capaz de realizar la adquisición de un EEG con mejores prestaciones y un coste inferior al de aquellos disponibles en el mercado.

Tras cumplir este objetivo y liberar los esquemáticos diseñados se sientan unas bases que cualquier persona o institución con menos recursos puede aprovechar para construir y mejorar su propio sistema. De esta forma se amplía sensiblemente el número de personas que puede realizar investigaciones relacionadas con el estudio de un EEG y, con ello, las posibilidades de llegar a conocer en un futuro el funcionamiento de nuestro cerebro.

La investigación en este campo a corto plazo puede ayudar a la detección de patrones que estén asociados a enfermedades o incluso podría permitir realizar interfaces cerebro-ordenador (BCI) que faciliten la vida a aquellas personas que sufren algún tipo de discapacidad motora o mental.

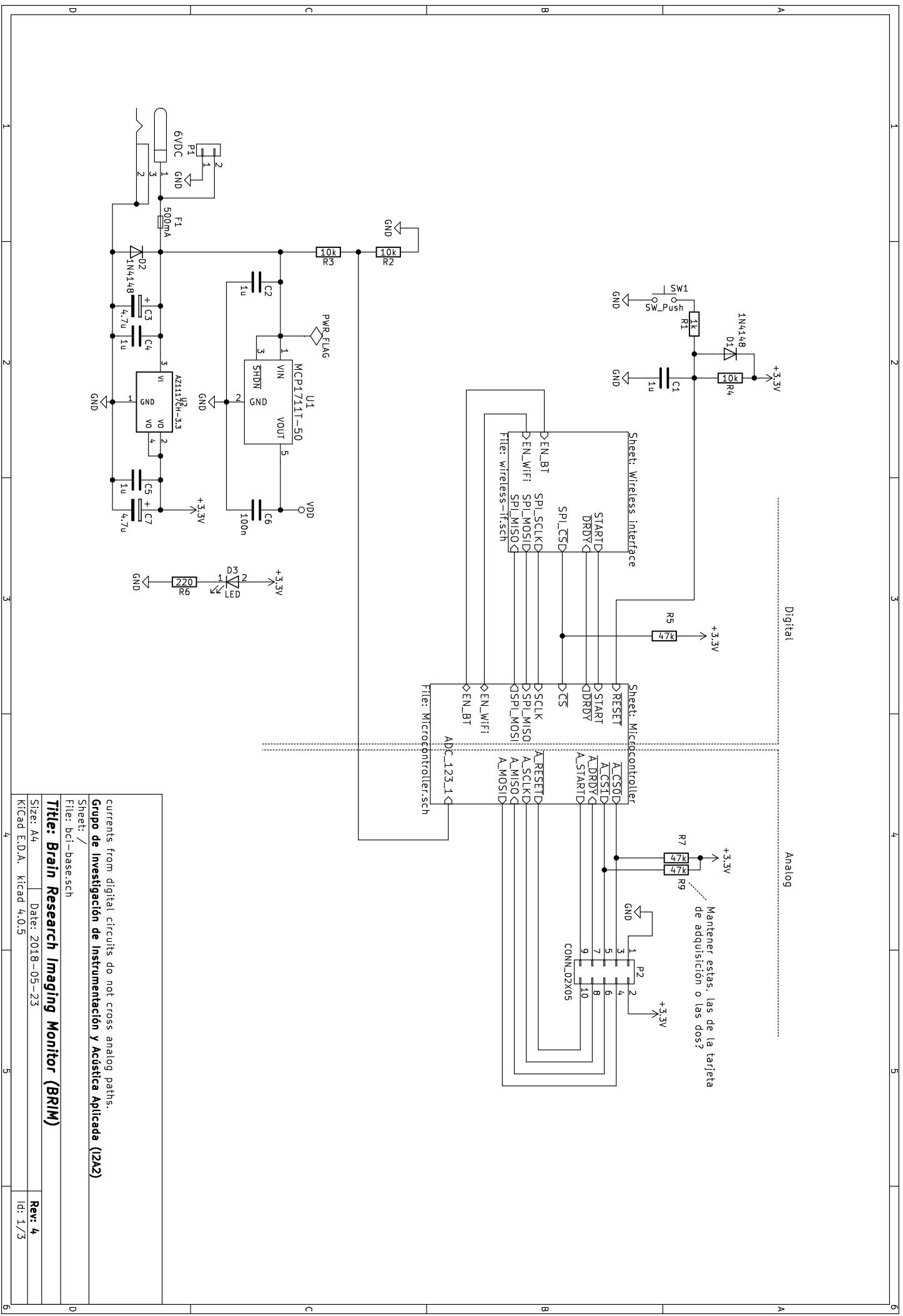
Todos los componentes electrónicos presentes en este proyecto cumplen con la directiva RoHS (*Restriction of Hazardous Substances*) por lo que se puede concluir que este proyecto cumple también con dicha directiva, es decir, las sustancias peligrosas comúnmente presentes en los componentes electrónicos se encuentran dentro de unos márgenes definidos por la Comunidad Europea.



Figura B.1: Directiva RoHS [35]

C

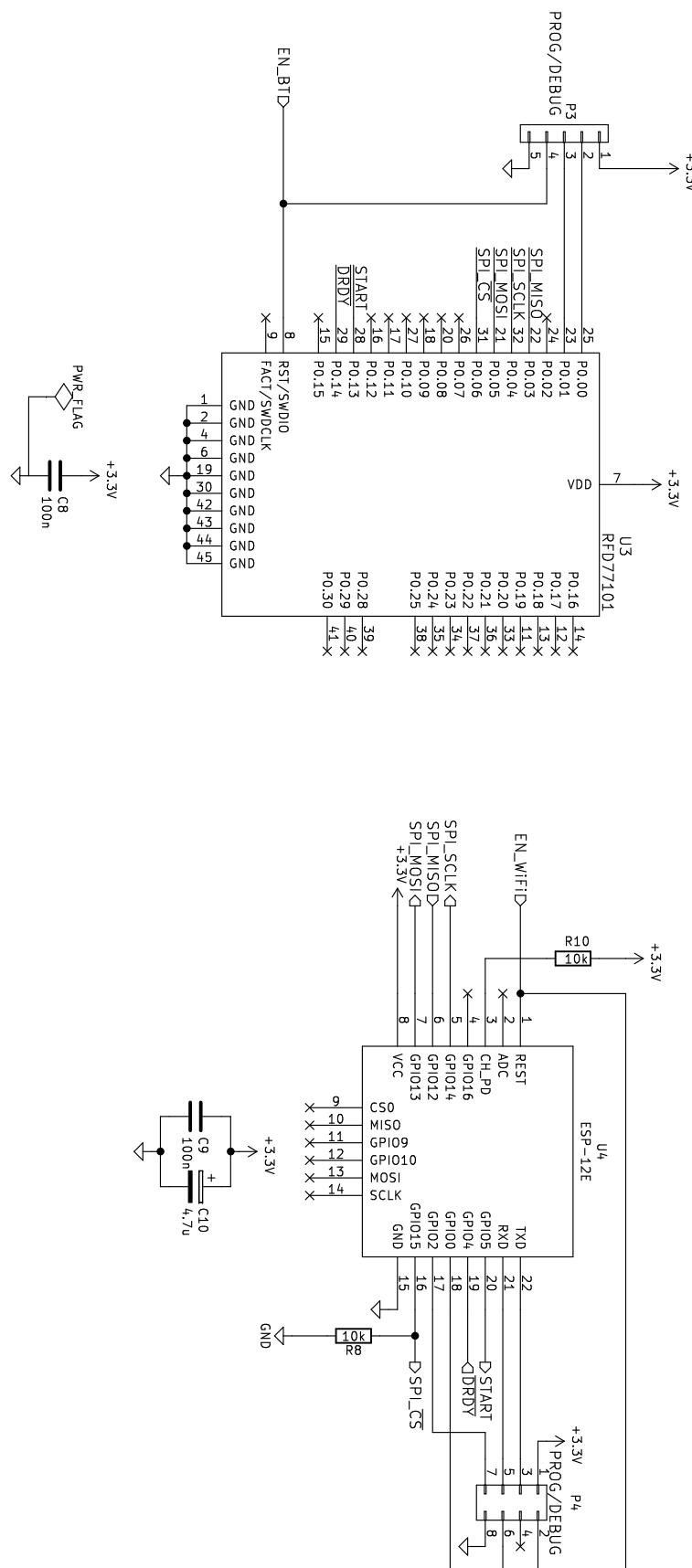
Esquemático del proyecto



Grupo de Investigación de Instrumentación y Acústica Aplicada (IIAZ)

Title: Brain Research Imaging Monitor (BRIM)

Size: A4	Date: 2018-05-23	Rev: 4
KICad E.D.A.	kicad 4.0.5	Id: 1/3
1	2	3
4	5	6



currents from digital circuits do not cross analog paths.

Grupo de Investigación de Instrumentación y Acústica Aplicada (IIA2)

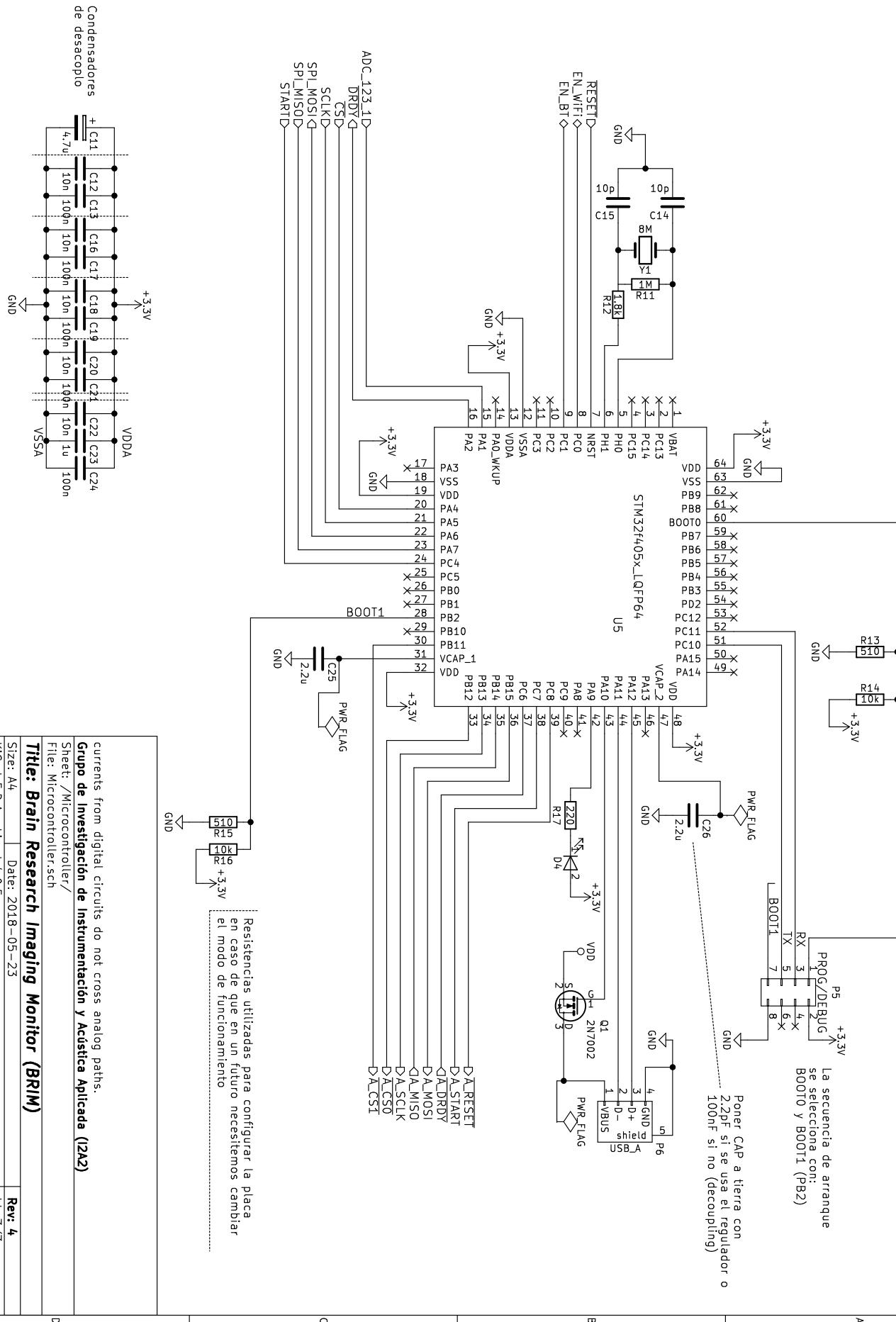
Sheet: /Wireless interface/

File: wireless-if.sch

Title: Brain Research Imaging Monitor (BRIM)

Size: A4	Date: 2018-05-23	Rev: 4
KICad E.D.A.	kicad 4.0.5	Id: 2/3
1	2	3
4	5	6

Resistencias utilizadas para configurar la placa
en caso de que en un futuro necesitemos cambiar
el modo de funcionamiento



currents from digital circuits do not cross analog paths.

Grupo de Investigación de Instrumentación y Acústica Aplicada (IIA2)

Sheet: /Microcontroller/
File: Microcontroller.sch

Title: Brain Research Imaging Monitor (BRIM)

Size: A4 Date: 2018-05-23

Rev. 4

Id: 3/3

6

D

Firmware del ESP12-E

Debido a la nueva normativa que permite la presentación de la memoria y documentación en formato digital se ha considerado oportuno la inclusión del código desarrollado para este proyecto en este documento, pues puede servir de referencia para futuros trabajos similares. Sin embargo, teniendo en cuenta su gran extensión, si se desea imprimir este documento se recomienda omitir la impresión de aquellos anexos relacionados con el código, especialmente el anexo E.

Este apéndice recoge el código desarrollado para el ESP12-E. Incluye todo lo necesario para hacer de interfaz entre el ordenador y el microcontrolador STM32F4.

```
0  /*
5
    GPIO      Name   |   STM
  -----
5  12        MISO   |   PA7
  13        MOSI   |   PA6
  14        SCK    |   PA5
  15        SS     |   PA4
10 */
11 #include <SPI.h>
12 #include <ESP8266WiFi.h>
13 #include <WiFiUdp.h>
14
15 // Funciones prototipo
16 uint8_t read_command(uint8_t command);
17 void FloatArrayFromSTM(float F_array [] , int N_elementos);
18 float floatFromSTM(void);
19 uint8_t String2int (String Array);
20 void Float2TCP (float Float_data , WiFiClient client);
```

```
char ssid [] = "SSID";
char pass [] = "PASSWORD";

25 IPAddress local_IP(192,168,4,22);
IPAddress gateway(192,168,4,9);
IPAddress subnet(255,255,255,0);
IPAddress remote_IP(192,168,4,121);

30 // Create an instance of the server
// specify the port to listen on as an argument
WiFiServer server(80);
WiFiUDP Udp;

35 unsigned int localPort = 2390;           // local port to listen on

// Definición de los pines
// Función GPIO#
40 #define HSPI_CS    15
#define HSPI_CLK   14
#define HSPI_MISO  12
#define HSPI_MOSI  13
#define DRDY_N     4
45 #define START      5
#define LED        2

#define hotspot false // hotspot = true
                  // wifi    = false
50 #define debug true

int i=1;

// la rutina de setup corre una vez o cuando se presiona reset
55 void setup() {
    Serial.begin(115200);
    SPI.begin();

    //SPI.beginTransaction(SPISettings(16000000, MSBFIRST, SPI_MODE1));
60
    SPI.setClockDivider(SPI_CLOCK_DIV2); //Divides 16MHz clock by 2 to set
    CLK speed to 4MHz
    SPI.setDataMode(SPI_MODE1); //clock polarity = 0; clock phase = 1 (pg.
    8)
    SPI.setBitOrder(MSBFIRST); //data format is MSB (pg. 25)

65 pinMode(HSPI_CS,OUTPUT);
pinMode(HSPI_CLK,SPECIAL);
pinMode(HSPI_MISO,SPECIAL);
pinMode(HSPI_MOSI,SPECIAL);
pinMode(DRDY_N,INPUT);
70 pinMode(START,OUTPUT);
pinMode(LED,OUTPUT);

    Serial.println();
```

```

75 #if (hotspot == false)

    // We start by connecting to a WiFi network
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, pass);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");

    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
90 #else

    Serial.print("Setting soft-AP configuration ... ");
    Serial.println(WiFi.softAPConfig(local_IP, gateway, subnet) ? "Ready" : "Failed!");

95 Serial.print("Setting soft-AP ... ");
    Serial.println(WiFi.softAP("ESPBCI_WIFI", "Password_01", false) ? "Ready" : "Failed!");

    Serial.print("Soft-AP IP address = ");
    Serial.println(WiFi.softAPIP());

100#endif

// Start the server
105 Serial.println("Starting servers");
    server.begin();
    Udp.begin(localPort);
    Serial.println("Server started");
    Serial.print("Local port: ");
    Serial.println(Udp.localPort());

110}

// la rutina loop corre constantemente
115 void loop() {
    uint8_t command2STM = 0x02;
    uint8_t response = 0x00;
    float F_data[8];
    float F_data_array[250];
120    int N_elementos = 250;
    long t1 = 0;
    int offset_configurar = 0x00;

125    //##### Server handler #####
    // Check if a client has connected

```

```
WiFiClient client = server.available();
if (!client) {
    //Serial.println("No client available");
    return;
}

// Wait until the client sends some data
130 Serial.println("new client");
t1 = millis();
while (!client.available()) {
    delay(1);
    Serial.print(".");
    ESP.wdtFeed(); // Evita reinicios por watchdog
140 if (millis() - t1 >= 10000) {
    Serial.println("Timeout");
    return;
}
145 Serial.println();

// Read the first line of the request
String req = client.readStringUntil('\r\n');

150 #if (debug == true)
    Serial.println(req);
#endif

155 // Match the request

    if (req.indexOf("Configurar") != -1)
    {
        command2STM = 0xF0;
160        Serial.println("Configurar");
    }
    else if (req.indexOf("Single float") != -1)
    {
        command2STM = 0x01;
165        Serial.println("Single float");
    }
    else if (req.indexOf("Float array") != -1)
    {
        command2STM = 0x02;
170        Serial.println("Float array");
    }
    else{
        Serial.println("invalid request");
        client.print("invalid request\r\n");
175        delay(1);
        client.flush();
        //client.stop();
        return;
    }
180 }
```

```

// Prepara la respuesta para LabView
String s = "";
//client.setNoDelay(true);
185
while (response == 0x00) // Espera a que el STM se sincronice con el ESP
{
    response = read_command(command2STM);
    ESP.wdtFeed(); // Evita reinicios por watchdog
190
    Serial.println(response,HEX);
}

#if (debug == true)
    Serial.print("Comando recibido: ");
195
    Serial.println(response,HEX);
#endif

switch (command2STM) {

200
    case 0xF0:
        #if (debug == true)
            Serial.println("CASO 0: Configurar");
        #endif
        offset_configurar = req.indexOf("Configurar");
        read_command(String2int(req.substring(offset_configurar+11,
205
            offset_configurar+19))); //Config 1
        read_command(String2int(req.substring(offset_configurar+20,
            offset_configurar+28))); //Config 2
        read_command(String2int(req.substring(offset_configurar+29,
            offset_configurar+37))); //Config 3
        read_command(String2int(req.substring(offset_configurar+38,
            offset_configurar+46))); //Config 4

210
        read_command(String2int(req.substring(offset_configurar+47,
            offset_configurar+55))); //Channel 1
        read_command(String2int(req.substring(offset_configurar+56,
            offset_configurar+64))); //Channel 2
        read_command(String2int(req.substring(offset_configurar+65,
            offset_configurar+73))); //Channel 3
        read_command(String2int(req.substring(offset_configurar+74,
            offset_configurar+82))); //Channel 4
        read_command(String2int(req.substring(offset_configurar+83,
            offset_configurar+91))); //Channel 5
215
        read_command(String2int(req.substring(offset_configurar+92,
            offset_configurar+100))); //Channel 6
        read_command(String2int(req.substring(offset_configurar+101,
            offset_configurar+109))); //Channel 7
        read_command(String2int(req.substring(offset_configurar+110,
            offset_configurar+118))); //Channel 8

220
        #if (debug == true)
            Serial.println("CASO 0: END");
        #endif

        break;
        case 0x01: // El STM va a transmitir un dato de tipo float
}

```

```
225      #if (debug == true)
        Serial.println("CASO 1: Single Float");
#endif
        for (int i = 0; i<8; i++)
        {
230          F_data[i] = floatFromSTM();
          Serial.println(F_data[i],7);
        }
        for (int i = 0; i<8; i++)
        {
235          Float2TCP(F_data[i], client);
        }
        client.write("\r\n"); // cierra la conexión con Labview

        #if (debug == true)
240          Serial.println("CASO 1: END");
        #endif
        break;
    case 0x02:
        #if (debug == true)
245          Serial.println("CASO 2: ARRAY");
        #endif
        t1 = millis();
        FloatArrayFromSTM(F_data_array, N_elementos);

250        for (i=0; i<250; i++)
        {
          Serial.println(F_data_array[i],7);
        }

255        for (i=0; i<250; i++)
        {
          Float2TCP(F_data_array[i], client);
        }
260        client.write("\r\n"); // cierra la conexión con Labview
        Serial.println("END - CASO 2: ARRAY");

        #if (debug == true)
265          Serial.println("CASO 2: END");
        #endif

        break;
    default:
270        #if (debug == true)
          Serial.println("Default");
        #endif
        ESP.wdtFeed(); // Evita reinicios por watchdog

        client.write("\r\n"); // cierra la conexión con Labview

        #if (debug == true)
275          Serial.println("Default: END");
        #endif
```

```

280      break;
    }

285     Serial.println("Client disconnected");
    client.flush();
}

290

295     uint8_t read_command(uint8_t command)
{
    while (digitalRead(DRDY_N) != LOW)
    {
        ESP.wdtFeed(); // Evita reinicios por watchdog
        //Serial.println("Waiting for DRDY");
300    }

    digitalWrite(HSPI_CS, LOW);
    command = SPI.transfer(command);
    digitalWrite(HSPI_CS, HIGH);
305    while(digitalRead(DRDY_N)==LOW){} // Bloquea hasta que el ESP considera
        terminada la transmisión

    return command;
}

310 void FloatArrayFromSTM(float F_array [], int N_elementos)
{
    union miDatos{
        struct
    {
315        byte b[4]; // Array de bytes de tamaño igual al tamaño de la
            primera variable: int = 2 bytes, float = 4 bytes
        } split;
        float fval;
    } F_recibido;
    F_recibido.fval = 8765.4321;
320    int i = 0;

    while (digitalRead(DRDY_N) != LOW)
    {
        ESP.wdtFeed(); // Evita reinicios por watchdog
325        //Serial.println("Waiting for DRDY");
    }

    long t1 = millis();
    digitalWrite(HSPI_CS, LOW);
330    for (i = 0; i < N_elementos; i++)
    {
        F_recibido.split.b[0] = SPI.transfer(0b01000000);
}

```

```
335         F_recibido.split.b[1] = SPI.transfer(0b01000000);
            F_recibido.split.b[2] = SPI.transfer(0b01000000);
            F_recibido.split.b[3] = SPI.transfer(0b01000000);

            F_array[i]=F_recibido.fval;
        }

340 Serial.println(millis()-t1);

        digitalWrite(HSPI_CS, HIGH);

345 while (digitalRead(DRDY_N)==LOW)
{
    ESP.wdtFeed(); // Evita reinicios por watchdog
}
// Espera a que DRDY levante, significa que se ha acabado el bucle.
// Bloquea el micro pero asegura que no entra más de una vez
350 if (digitalRead(LED)==1)
    digitalWrite(LED,LOW);
else
    digitalWrite(LED,HIGH);
355 }

float floatFromSTM(void)
{
    union miDato{
        struct
        {
            byte b[4]; // Array de bytes de tamaño igual al tamaño de la
            primera variable: int = 2 bytes, float = 4 bytes
        }split;
        float fval;
    } F_recibido;

365     F_recibido.fval = 8765.4321;
     int i = 0;

370     digitalWrite(HSPI_CS, LOW);
     while (digitalRead(DRDY_N)!=LOW)
    {
        ESP.wdtFeed(); // Evita reinicios por watchdog
        //Serial.println("Waiting for DRDY");
375    }
    F_recibido.split.b[0] = SPI.transfer(0b01000000);
    F_recibido.split.b[1] = SPI.transfer(0b01000000);
    F_recibido.split.b[2] = SPI.transfer(0b01000000);
    F_recibido.split.b[3] = SPI.transfer(0b01000000);

380     digitalWrite(HSPI_CS, HIGH);

     while (digitalRead(DRDY_N)==LOW)
    {
        ESP.wdtFeed(); // Evita reinicios por watchdog
385    }
```

```

// Espera a que DRDY levante, significa que se ha acabado el bucle.
// Bloquea el micro pero asegura que no entra más de una vez

390 if (digitalRead(LED)==1)
    digitalWrite(LED,LOW);
else
    digitalWrite(LED,HIGH);

395 return F_recibido.fval;
}

400 uint8_t String2int (String Array)
{
    uint8_t output = 0x00;
    for (int i = 0; i < 8; i++)
        bitWrite(output, i, bitRead(Array[7-i], 0));

    return output;
405 }

void Float2TCP (float Float_data, WiFiClient client)
{
    union miDatos{
        struct
        {
            byte b[4];      // Array de bytes de tamaño igual al tamaño de la
            // primera variable: int = 2 bytes, float = 4 bytes
        } split;
        float fval;
415 } F_recibido;

    F_recibido.fval = Float_data;

    client.write(F_recibido.split.b[3]); // Envía la respuesta a LabView
    client.write(F_recibido.split.b[2]); // Envía la respuesta a LabView
    client.write(F_recibido.split.b[1]); // Envía la respuesta a LabView
    client.write(F_recibido.split.b[0]); // Envía la respuesta a LabView*/
}

```

Código D.1: ESP:master_fast.ino

E

Firmware del STM32F4

A continuación se incluye el firmware desarrollado para el microcontrolador STM32F4, mostrando los ficheros más importantes y obviando aquellos generados de forma automática o presentes en las librerías aunque estos hayan sido de utilidad.

```
0  /*
*****  
* File Name          : main.c  
* Description       : Main program body  
*****  
5   * This notice applies to any and all portions of this file  
* that are not between comment pairs USER CODE BEGIN and  
* USER CODE END. Other portions of this file , whether  
* inserted by the user or by software development tools  
* are owned by their respective copyright owners.  
10  *  
* Copyright (c) 2018 STMicroelectronics International N.V.  
* All rights reserved.  
*  
* Redistribution and use in source and binary forms, with or without  
15  * modification , are permitted , provided that the following conditions are  
* met:  
*  
* 1. Redistribution of source code must retain the above copyright notice  
*    ,  
*    this list of conditions and the following disclaimer.  
* 2. Redistributions in binary form must reproduce the above copyright  
*    notice ,  
20  *    this list of conditions and the following disclaimer in the  
*    documentation  
*    and/or other materials provided with the distribution.  
* 3. Neither the name of STMicroelectronics nor the names of other  
*    contributors to this software may be used to endorse or promote
```

```
products
*      derived from this software without specific written permission.
25 * 4. This software, including modifications and/or derivative works of
*      this
*      software, must execute solely and exclusively on microcontroller or
*      microprocessor devices manufactured by or for STMicroelectronics.
* 5. Redistribution and use of this software other than as permitted
under
*      this license is void and will automatically terminate your rights
under
30 *      this license.
*
* THIS SOFTWARE IS PROVIDED BY STMICROELECTRONICS AND CONTRIBUTORS "AS IS"
"
* AND ANY EXPRESS, IMPLIED OR STATUTORY WARRANTIES, INCLUDING, BUT NOT
* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
35 * PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL
PROPERTY
* RIGHTS ARE DISCLAIMED TO THE FULLEST EXTENT PERMITTED BY LAW. IN NO
EVENT
* SHALL STMICROELECTRONICS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,
INDIRECT,
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
BUT NOT
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA,
40 * OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY
OF
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE,
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*
45 ****
*/
/* Includes _____ */
#include "main.h"
50 #include "stm32f4xx_hal.h"
#include "usb_host.h"
/* USER CODE BEGIN Includes */

#include "util.h"
55 #include "ADS1299.h"
#include "stdbool.h"
#include <stdlib.h>

// Includes y defines relacionados con el filtro FIR
60 #include "arm_math.h"
#include "math_helper.h"

/* USER CODE END Includes */

65 /* Private variables _____ */
ADC_HandleTypeDef hadc1;
```

```

SPI_HandleTypeDef hspi1;
SPI_HandleTypeDef hspi2;
70 UART_HandleTypeDef huart4;

/* USER CODE BEGIN PV */
/* Private variables _____*/
75 /* USER CODE END PV */

/* Private function prototypes _____*/
void SystemClock_Config(void);
80 static void MX_GPIO_Init(void);
static void MX_SPI1_Init(void);
static void MX_SPI2_Init(void);
static void MX_ADC1_Init(void);
static void MX_UART4_Init(void);
85 void MX_USB_HOST_Process(void);

/* USER CODE BEGIN PFP */
/* Private function prototypes _____*/
90 uint8_t commandFromESP (uint8_t command, SPI_HandleTypeDef *SPI);
void float2ESP (float32_t data2ESP, SPI_HandleTypeDef *SPI);
void floatArray2ESP ( float32_t data2ESP_array[], int N_elementos,
    SPI_HandleTypeDef *SPI, UART_HandleTypeDef *huart4);

/* USER CODE END PFP */
95 /* USER CODE BEGIN 0 */

// global variables
unsigned long blink_interval_millis;
100 float32_t channel_1 [LENGTH_SAMPLES];
float32_t channel_2 [LENGTH_SAMPLES];
float32_t channel_3 [LENGTH_SAMPLES];
float32_t channel_4 [LENGTH_SAMPLES];
float32_t channel_5 [LENGTH_SAMPLES];
105 float32_t channel_6 [LENGTH_SAMPLES];
float32_t channel_7 [LENGTH_SAMPLES];
float32_t channel_8 [LENGTH_SAMPLES];

bool wait = false;
110 uint8_t bucle=1;
uint8_t k = 0;
int i canal=1;
int i;
bool acabar = false;
115 bool Data_ready = true;
uint8_t comando_temp = 0x00;

uint8_t data[27];
120 uint8_t command = 0x00;

```

```
int channel = 1;

float32_t data2ESP = 1234.5678;
125
bool shared_negative_electrode = true;

/*
 * Declare Test output buffer
 */
130 static float32_t testOutput[LENGTH_SAMPLES];
/*
 * Declare State buffer of size (numTaps + blockSize - 1)
 */
135 static float32_t firStateF32[BLOCK_SIZE + NUM_TAPS - 1];
/*
 ** FIR Coefficients buffer generated using fir1() MATLAB function.
 ** fir1(28, 6/24)
 */
140 const float32_t firCoeffs32[NUM_TAPS] = { -0.0138029831773458,
    0.0395399929599544, 0.0428349923732840, -0.0175674331348036,
    -0.0605461382912526, -0.0197781448960783, 0.0543574323501128,
    0.0566917462875556, -0.0224447398190889, -0.0748391427030912,
    -0.0236934079009084, 0.0631991587299459, 0.0640421490722732,
    -0.0246563468927641, 0.9200000000000000, -0.0246563468927641,
    0.0640421490722732, 0.0631991587299459, -0.0236934079009084,
    -0.0748391427030912, -0.0224447398190889, 0.0566917462875556,
    0.0543574323501128, -0.0197781448960783, -0.0605461382912526,
    -0.0175674331348036, 0.0428349923732840, 0.0395399929599544,
    -0.0138029831773458};

/*
 * Global variables for FIR LPF Example
 */
145 uint32_t blockSize = BLOCK_SIZE;
uint32_t numBlocks = LENGTH_SAMPLES/BLOCK_SIZE;
float32_t snr;

// Inicialización de todas las variables relacionadas con el filtrado
150 int i;
arm_fir_instance_f32 S;
arm_status status;
float32_t *inputF32 = &channel_1[0]; // Definición del puntero de
// la señal a filtrar.
float32_t *outputF32 = &testOutput[0]; // Definición del puntero de
// la señal filtrada.

/* USER CODE END 0 */

int main(void)
{
    /* USER CODE BEGIN 1 */


```

```

165  /* USER CODE END 1 */

166  /* MCU Configuration
167   * -----*/

168  /* Reset of all peripherals , Initializes the Flash interface and the
169   * Systick. */
170  HAL_Init();

171  /* USER CODE BEGIN Init */

172  /* USER CODE END Init */

173  /* Configure the system clock */
174  SystemClock_Config();

175  /* USER CODE BEGIN SysInit */

176  /* USER CODE END SysInit */

177  /* Initialize all configured peripherals */
178  MX_GPIO_Init();
179  MX_SPI1_Init();
180  MX_SPI2_Init();
181  MX_ADC1_Init();
182  MX_UART4_Init();
183  MX_USB_HOST_Init();

184  /* USER CODE BEGIN 2 */

185  // Inicialización del estado de ciertos pines
186  HAL_GPIO_WritePin(DRDY_N_GPIO_Port, DRDY_N_Pin, GPIO_PIN_SET);
187  HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);
188  HAL_GPIO_WritePin(A_START_GPIO_Port, A_START_Pin, GPIO_PIN_SET);

189  // Reiniciamos el ESP12-E y lo dejamos habilitado

190  HAL_GPIO_WritePin(EN_WIFI_GPIO_Port, EN_WIFI_Pin, GPIO_PIN_RESET);
191  HAL_Delay(100);
192  HAL_GPIO_WritePin(EN_WIFI_GPIO_Port, EN_WIFI_Pin, GPIO_PIN_SET);

193  // Parpadeo del LED del ADS y el STM para comprobar si han arrancado bien

194  HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_RESET);    //LED ON
195  adc_wreg(GPIO, 0x1C, &hspi2);          // Led 1 on, led 2 off
196  HAL_Delay(500);
197  HAL_GPIO_WritePin(LED_GPIO_Port, LED_Pin, GPIO_PIN_SET);    //LED OFF
198  adc_wreg(GPIO, 0x2C, &hspi2);          // Led 1 on, led 2 off
199  HAL_Delay(500);

200  // Reiniciamos el ADS y lo dejamos habilitado

```

```
HAL_GPIO_WritePin(A_RESET_N_GPIO_Port, A_RESET_N_Pin, GPIO_PIN_RESET) ;
HAL_Delay(100) ;
HAL_GPIO_WritePin(A_RESET_N_GPIO_Port, A_RESET_N_Pin, GPIO_PIN_SET) ;

220   uint8_t RESET_opcode = 0x06 ;
      uint8_t zero = 0x00 ;

HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_RESET) ;
225   HAL_SPI_TransmitReceive(&hspi2, &RESET_opcode, &zero, 1, 100) ;
HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_SET) ;
HAL_Delay(1000) ;

// Initial configuration of the ADS
230   uint8_t config [4];
      uint8_t config_channel [8];

config[0] = CONFIG1_reserved | DR_250_SPS;
config[1] = CONFIG2_reserved | INT_CAL | CAL_FREQ_SLOW;
235   config[2] = PD_REFBUF | CONFIG3_reserved | BIASREF_INT | PD_BIAS;
config[3] = PD_LOFF_COMP;

config_channel [0] = TEST_SIGNAL | GAIN_24X;
config_channel [1] = TEST_SIGNAL | GAIN_24X;
240   config_channel [2] = TEST_SIGNAL | GAIN_24X;
config_channel [3] = TEST_SIGNAL | GAIN_24X;
config_channel [4] = TEST_SIGNAL | GAIN_24X;
config_channel [5] = TEST_SIGNAL | GAIN_24X;
config_channel [6] = TEST_SIGNAL | GAIN_24X;
245   config_channel [7] = TEST_SIGNAL | GAIN_24X;

configADS(config, config_channel, &hspi2, &huart4);

uint8_t gain[8] = {0, 0, 0, 0, 0, 0, 0, 0};
250   for (int i = 0; i<8; i++)
{
    gain[i] = calcular_ganancia(config_channel[i]);
}

255 //=====

/* USER CODE END 2 */

260 /* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
265   uint8_t command2ESP = 0x02;
      // Begin of the states machine

      // Wait for command from ESP
      while (command == 0x00)
    {
270     command = commandFromESP(command2ESP, &hspi1);
```

```

    }

275   switch (command){
      case (0xF0): // Cambio en la configuración
        for (int i=0; i<4; i++){
          config[ i ] = commandFromESP(command, &hspi1 );
        }
        for (int i=0; i<8; i++){
          config_channel[ i ] = commandFromESP(command, &hspi1 );
        }
        configADS(config, config_channel, &hspi2, &huart4);

        for (int i = 0; i<8; i++)
        {
          gain[ i ] = calcular_ganancia(config_channel[ i ]);
        }

        command = 0x00;
290
      break;

      case (0x01): // Lectura de un dato y envío al ESP
295
        adquire_single_data(data, &hspi2, &huart4);
        // one_shot(data, &hspi2, &huart4);

        //adc_wreg(GPIO, 0x2C, &hspi2); // Led 1 on, led 2 off
300
        for (int i = 1; i<=8; i++)
        {
          data2ESP = byte2float(data[ i*3 ], data[ i*3 +1], data[ i*3 + 2 ], gain[ i ]);
        }
        float2ESP (data2ESP, &hspi1);
305
        //adc_wreg(GPIO, 0x1C,&hspi2); // Led 1 off, led 2 on

        Serial.println_N(data[3], &huart4);
        Serial.println_N(data[4], &huart4);
310
        Serial.println_N(data[5], &huart4);

        command = 0x00;

      break;
315
      case (0x02): // Lectura continua de datos y envío al ESP
320
        adquire_array_data (data, channel_1, channel_2, channel_3, channel_4,
        channel_5, channel_6, channel_7, channel_8, gain, &hspi2, &huart4);
        // _____
        // Call the FIR process function for every blockSize samples
        // _____
        for (i=0; i < (BLOCK_SIZE + NUM_TAPS - 1); i++)

```

```
325     {
5         firStateF32 [ i ] = channel_1 [ i ];
6     }
7     // Call FIR init function to initialize the instance structure.
8     arm_fir_init_f32 (&S, NUM_TAPS, (float32_t *)&firCoeffs32 [ 0 ], &
9     firStateF32 [ 0 ], blockSize );
10
11    for ( i = 0; i < numBlocks; i ++ ) //
12    {
13        arm_fir_f32 (&S, inputF32 + ( i * blockSize ), outputF32 + ( i * 
14 blockSize ), blockSize );
15    }
16
17    // one_shot_array ( data , channel_1 , 1 , &hspi2 , &huart4 );
18
19    floatArray2ESP ( testOutput , LENGTH_SAMPLES, &hspi1 , &huart4 );
20    //floatArray2ESP ( channel_1 , LENGTH_SAMPLES, &hspi1 , &huart4 );
21    command = 0x00 ;
22
23    break ;
24
25    default :
26
27        command = 0x00 ;
28
29        break ;
30    }
31
32
33    /* USER CODE END WHILE */
34    MX_USB_HOST_Process () ;
35
36    /* USER CODE BEGIN 3 */
37
38    }
39    /* USER CODE END 3 */
40}
41
42/** System Clock Configuration
43*/
44void SystemClock_Config ( void )
45{
46
47    RCC_OscInitTypeDef RCC_OscInitStruct ;
48    RCC_ClkInitTypeDef RCC_ClkInitStruct ;
49
50
51    /**Configure the main internal regulator output voltage
52    */
53    __HAL_RCC_PWR_CLK_ENABLE () ;
54
55    __HAL_PWR_VOLTAGESCALING_CONFIG ( PWR_REGULATOR_VOLTAGE_SCALE1 ) ;
56
57    /**Initializes the CPU, AHB and APB busses clocks
58    */
59
```

```

RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
RCC_OscInitStruct.HSEState = RCC_HSE_ON;
380 RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
RCC_OscInitStruct.PLL.PLLM = 8;
RCC_OscInitStruct.PLL.PLLN = 336;
RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
385 RCC_OscInitStruct.PLL.PLLQ = 7;
if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
{
    _Error_Handler(__FILE__, __LINE__);
}
390
    /* Initializes the CPU, AHB and APB busses clocks
     */
RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
395           |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

400 if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
{
    _Error_Handler(__FILE__, __LINE__);
}

405
    /* Enables the Clock Security System
     */
HAL_RCC_EnableCSS();

410
    /* Configure the Systick interrupt time
     */
HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq() / 1000);

415
    /* Configure the Systick
     */
HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

420
    /* SysTick_IRQn interrupt configuration */
    HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);
}

425
/* ADC1 init function */
static void MX_ADC1_Init(void)
{
    ADC_ChannelConfTypeDef sConfig;

        /* Configure the global features of the ADC (Clock, Resolution, Data
         Alignment and number of conversion)
         */
hadc1.Instance = ADC1;
430 hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
hadc1.Init.Resolution = ADC_RESOLUTION_12B;

```

```
435     hadc1.Init.ScanConvMode = DISABLE;
        hadc1.Init.ContinuousConvMode = DISABLE;
        hadc1.Init.DiscontinuousConvMode = DISABLE;
        hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
        hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
        hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
        hadc1.Init.NbrOfConversion = 1;
        hadc1.Init.DMAContinuousRequests = DISABLE;
440     hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
        if (HAL_ADC_Init(&hadc1) != HAL_OK)
        {
            _Error_Handler(__FILE__, __LINE__);
        }
445     /* Configure for the selected ADC regular channel its corresponding
       rank in the sequencer and its sample time.
       */
450     sConfig.Channel = ADC_CHANNEL_1;
     sConfig.Rank = 1;
     sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
     if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
     {
         _Error_Handler(__FILE__, __LINE__);
     }
455 }
460 /* SPI1 init function */
static void MX_SPI1_Init( void )
{
    /* SPI1 parameter configuration*/
465     hspi1.Instance = SPI1;
     hspi1.Init.Mode = SPI_MODE_SLAVE;
     hspi1.Init.Direction = SPI_DIRECTION_2LINES;
     hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
     hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
     hspi1.Init.CLKPhase = SPI_PHASE_2EDGE;
     hspi1.Init.NSS = SPI_NSS_HARD_INPUT;
470     hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
     hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
     hspi1.Init.CRCCalculation = SPI_CRCALCULATION_DISABLE;
     hspi1.Init.CRCPolynomial = 10;
     if (HAL_SPI_Init(&hspi1) != HAL_OK)
475     {
         _Error_Handler(__FILE__, __LINE__);
     }
}
480 /* SPI2 init function */
static void MX_SPI2_Init( void )
{
    /* SPI2 parameter configuration*/
485 }
```

```

    hspi2.Instance = SPI2;
    hspi2.Init.Mode = SPI_MODE_MASTER;
    hspi2.Init.Direction = SPI_DIRECTION_2LINES;
    hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
490   hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
    hspi2.Init.CLKPhase = SPI_PHASE_2EDGE;
    hspi2.Init.NSS = SPI_NSS_SOFT;
    hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
    hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
495   hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
    hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
    hspi2.Init.CRCPolynomial = 10;
    if (HAL_SPI_Init(&hspi2) != HAL_OK)
    {
500     _Error_Handler(__FILE__, __LINE__);
    }

}

505 /* UART4 init function */
static void MX_UART4_Init(void)
{
    huart4.Instance = UART4;
510   huart4.Init.BaudRate = 115200;
    huart4.Init.WordLength = UART_WORDLENGTH_8B;
    huart4.Init.StopBits = UART_STOPBITS_1;
    huart4.Init.Parity = UART_PARITY_NONE;
    huart4.Init.Mode = UART_MODE_TX_RX;
515   huart4.Init.HwFlowCtl = UART_HWCONTROL_NONE;
    huart4.Init.OverSampling = UART_OVERSAMPLING_16;
    if (HAL_UART_Init(&huart4) != HAL_OK)
    {
520     _Error_Handler(__FILE__, __LINE__);
    }

}

525 /** Configure pins as
 * Analog
 * Input
 * Output
 * EVENT_OUT
 * EXTI
530 */
static void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct;
535
    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOH_CLK_ENABLE();
    __HAL_RCC_GPIOC_CLK_ENABLE();
    __HAL_RCC_GPIOA_CLK_ENABLE();
    __HAL_RCC_GPIOB_CLK_ENABLE();
540
}

```

```
/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOC, EN_WIFI_Pin|A_START_Pin|A_RESET_N_Pin,
                  GPIO_PIN_SET);

545 /*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(EN_BT_GPIO_Port, EN_BT_Pin, GPIO_PIN_RESET);

/*Configure GPIO pin Output Level */
550 HAL_GPIO_WritePin(GPIOA, DRDY_N_Pin|LED_Pin, GPIO_PIN_SET);

/*Configure GPIO pin Output Level */
555 HAL_GPIO_WritePin(A_CS1_N_GPIO_Port, A_CS1_N_Pin, GPIO_PIN_RESET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_SET);

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(EN_USB_GPIO_Port, EN_USB_Pin, GPIO_PIN_RESET);

560 /*Configure GPIO pins : EN_WIFI_Pin EN_BT_Pin A_START_Pin A_RESET_N_Pin
 */
GPIO_InitStruct.Pin = EN_WIFI_Pin|EN_BT_Pin|A_START_Pin|A_RESET_N_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
565 HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/*Configure GPIO pins : DRDY_N_Pin LED_Pin EN_USB_Pin */
GPIO_InitStruct.Pin = DRDY_N_Pin|LED_Pin|EN_USB_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
570 GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pins : START_Pin A_DRDY_N_Pin */
575 GPIO_InitStruct.Pin = START_Pin|A_DRDY_N_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

580 /*Configure GPIO pins : A_CS1_N_Pin A_CS0_N_Pin */
GPIO_InitStruct.Pin = A_CS1_N_Pin|A_CS0_N_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
585 HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

590 uint8_t commandFromESP(uint8_t command, SPI_HandleTypeDef *hspi1)
{
    uint8_t response = 0x00;
```

```

595     HAL_GPIO_WritePin(DRDY_N_GPIO_Port, DRDY_N_Pin, GPIO_PIN_RESET);
      HAL_SPI_TransmitReceive(hspi1, &command, &response, 1, 100);
      HAL_GPIO_WritePin(DRDY_N_GPIO_Port, DRDY_N_Pin, GPIO_PIN_SET);
      HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin); //LED OFF
      HAL_Delay(50);

600     return response;
}

605 void float2ESP (float32_t data2ESP, SPI_HandleTypeDef *SPI)
{
    union miDatos{
        struct
        {
            uint8_t b[4]; // Array de bytes de tamaño igual al tamaño de la primera variable: int = 2 bytes, float = 4 bytes
            }split;
            float32_t fval;
        } float_data;

610     float_data.fval=data2ESP;

615     HAL_GPIO_WritePin(DRDY_N_GPIO_Port, DRDY_N_Pin, GPIO_PIN_RESET);
      HAL_SPI_Transmit(SPI, float_data.split.b, 4, 100);
      HAL_GPIO_WritePin(DRDY_N_GPIO_Port, DRDY_N_Pin, GPIO_PIN_SET);
      HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin); //LED OFF
      HAL_Delay(10);

620 }

625 void floatArray2ESP (float32_t data2ESP_array[], int N_elementos,
                      SPI_HandleTypeDef *SPI, UART_HandleTypeDef *huart4)
{
    union miDatos{
        struct
        {
            uint8_t b[4]; // Array de bytes de tamaño igual al tamaño de la primera variable: int = 2 bytes, float = 4 bytes
            }split;
            float32_t fval;
        } float_data;

630     uint8_t buffer [1000]; //tamaño máximo del buffer a usar
     int i = 0;

635     for (i = 0; i<N_elementos; i++)
    {
        float_data.fval = data2ESP_array[i];
        buffer[i*4] = float_data.split.b[0];
        buffer[i*4+1] = float_data.split.b[1];
        buffer[i*4+2] = float_data.split.b[2];
        buffer[i*4+3] = float_data.split.b[3];
    }

640     HAL_GPIO_WritePin(DRDY_N_GPIO_Port, DRDY_N_Pin, GPIO_PIN_RESET);
}

```

```
HAL_SPI_Transmit(SPI, buffer, 1000, 50);
HAL_GPIO_WritePin(DRDY_N_GPIO_Port,DRDY_N_Pin, GPIO_PIN_SET);
HAL_GPIO_TogglePin(LED_GPIO_Port,LED_Pin); //LED OFF
650 }
/* USER CODE END 4 */

/**
 * @brief This function is executed in case of error occurrence.
 * @param None
 * @retval None
 */
void _Error_Handler(char * file , int line)
660 {
    /* USER CODE BEGIN Error_Handler_Debug */
    /* User can add his own implementation to report the HAL error return
       state */
    while(1)
    {
    }
665 /* USER CODE END Error_Handler_Debug */
}

#ifndef USE_FULL_ASSERT
670 /**
     * @brief Reports the name of the source file and the source line number
     * where the assert_param error has occurred.
     * @param file: pointer to the source file name
     * @param line: assert_param error line source number
     * @retval None
     */
void assert_failed(uint8_t* file , uint32_t line)
{
    /* USER CODE BEGIN 6 */
    /* User can add his own implementation to report the file name and line
       number,
       ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line
       ) */
    /* USER CODE END 6 */
680 }

685 }

#endif

690 /**
 * @}
 */
695 /**
 * @}
 */
696 **** (C) COPYRIGHT STMicroelectronics *****END OF FILE
```

****/

Código E.1: STM32F4:main.c

```

0  /**
*****  

* File Name      : main.hpp  

* Description     : This file contains the common defines of the  

*                   application  

*****  

5   * This notice applies to any and all portions of this file  

* that are not between comment pairs USER CODE BEGIN and  

* USER CODE END. Other portions of this file , whether  

* inserted by the user or by software development tools  

* are owned by their respective copyright owners.  

10  *  

* Copyright (c) 2018 STMicroelectronics International N.V.  

* All rights reserved.  

*  

* Redistribution and use in source and binary forms, with or without  

15  * modification , are permitted, provided that the following conditions are  

* met:  

*  

* 1. Redistribution of source code must retain the above copyright notice  

*,  

*   this list of conditions and the following disclaimer.  

* 2. Redistributions in binary form must reproduce the above copyright  

*   notice ,  

20  *   this list of conditions and the following disclaimer in the  

*   documentation  

*   and/or other materials provided with the distribution.  

* 3. Neither the name of STMicroelectronics nor the names of other  

*   contributors to this software may be used to endorse or promote  

*   products  

*   derived from this software without specific written permission.  

25  * 4. This software , including modifications and/or derivative works of  

*   this  

*   software , must execute solely and exclusively on microcontroller or  

*   microprocessor devices manufactured by or for STMicroelectronics.  

* 5. Redistribution and use of this software other than as permitted  

*   under  

*   this license is void and will automatically terminate your rights  

*   under  

30  *   this license.  

*  

* THIS SOFTWARE IS PROVIDED BY STMICROELECTRONICS AND CONTRIBUTORS "AS IS  

*"  

* AND ANY EXPRESS, IMPLIED OR STATUTORY WARRANTIES, INCLUDING, BUT NOT  

* LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A  

35  * PARTICULAR PURPOSE AND NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL  

* PROPERTY  

* RIGHTS ARE DISCLAIMED TO THE FULLEST EXTENT PERMITTED BY LAW. IN NO  

* EVENT  

* SHALL STMICROELECTRONICS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT,  

* INDIRECT,

```

```
* INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,  
40 BUT NOT  
* LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,  
DATA,  
* OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY  
OF  
* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING  
* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS  
SOFTWARE,  
* EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  
*  
45 ****  
*/  
/* Define to prevent recursive inclusion */  
#ifndef __MAIN_H  
#define __MAIN_H  
50 /* Includes */  
/* Includes */  
/* USER CODE BEGIN Includes */  
55 /* USER CODE END Includes */  
/* Private define */  
/*  
60 #define EN_WIFI_Pin GPIO_PIN_0  
#define EN_WIFI_GPIO_Port GPIOC  
#define EN_BT_Pin GPIO_PIN_1  
#define EN_BT_GPIO_Port GPIOC  
#define DRDY_N_Pin GPIO_PIN_2  
#define DRDY_N_GPIO_Port GPIOA  
65 #define START_Pin GPIO_PIN_4  
#define START_GPIO_Port GPIOC  
#define A_CS1_N_Pin GPIO_PIN_11  
#define A_CS1_N_GPIO_Port GPIOB  
#define A_CS0_N_Pin GPIO_PIN_12  
70 #define A_CS0_N_GPIO_Port GPIOB  
#define A_SCK_Pin GPIO_PIN_13  
#define A_SCK_GPIO_Port GPIOB  
#define A_MISO_Pin GPIO_PIN_14  
#define A_MISO_GPIO_Port GPIOB  
75 #define A_MOSI_Pin GPIO_PIN_15  
#define A_MOSI_GPIO_Port GPIOB  
#define A_DRDY_N_Pin GPIO_PIN_6  
#define A_DRDY_N_GPIO_Port GPIOC  
#define A_START_Pin GPIO_PIN_7  
80 #define A_START_GPIO_Port GPIOC  
#define A_RESET_N_Pin GPIO_PIN_8  
#define A_RESET_N_GPIO_Port GPIOC  
#define LED_Pin GPIO_PIN_9  
#define LED_GPIO_Port GPIOA
```

```

85 #define EN_USB_Pin GPIO_PIN_10
#define EN_USB_GPIO_Port GPIOA

/* ##### Assert Selection #####
*/
90 /**
 * @brief Uncomment the line below to expand the "assert_param" macro in
 *        the
 *        HAL drivers code
 */
/* #define USE_FULL_ASSERT      1U */

95 /* USER CODE BEGIN Private defines */

#define BLINK_INTERVAL_SETUP 100;
#define BLINK_INTERVAL_WAITING 500;
#define BLINK_INTERVAL_SENDING 2000;

100 // Defines relacionados con el filtro FIR
#define LENGTH_SAMPLES 250
#define SNR_THRESHOLD_F32 140.0f
#define BLOCK_SIZE 32
105 #define NUM_TAPS 29

/* USER CODE END Private defines */

110 #ifdef __cplusplus
extern "C" {
#endif
void _Error_Handler(char *, int);

#define Error_Handler() _Error_Handler(__FILE__, __LINE__)
115 #ifdef __cplusplus
}
#endif

120 /**
 * @}
 */
125 /**
 * @}
*/
126 #endif /* __MAIN_H */
/***** (C) COPYRIGHT STMicroelectronics *****END OF FILE ****/

```

Código E.2: STM32F4:main.h

```

0 /**
*****
* File Name          : ADS1299.c
* Description       : Function to interact with the ADS1299
*****
5 */

```

```
/* Includes */  
#include "ADS1299.h"  
10 #include "main.h"  
#include "util.h"  
#include "stdbool.h"  
#include <stdlib.h>  
  
15 void Blinky(SPI_HandleTypeDef *hspi2)  
{  
  
    uint8_t opcode_1 = WREG | GPIO; //0x54 (WREG + GPIO) Registro  
    encargado de escribir la configuración de los GPIO  
    uint8_t opcode_2 = 0x00; //Registros a leer (N-1)  
20    uint8_t reg_leido = 0x00;  
    uint8_t reg_LED_ON = 0x00;  
    uint8_t reg_LED_OFF = 0xF0;  
  
25    HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_RESET);  
    HAL_SPI_TransmitReceive(hspi2, &opcode_1, &reg_leido, 1, HAL_MAX_DELAY);  
    ;  
    HAL_SPI_TransmitReceive(hspi2, &opcode_2, &reg_leido, 1, HAL_MAX_DELAY);  
    ;  
    HAL_SPI_TransmitReceive(hspi2, &reg_LED_ON, &reg_leido, 1,  
    HAL_MAX_DELAY); //LED ON  
    HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_SET);  
30    HAL_Delay(500);  
  
    HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_RESET);  
    HAL_SPI_TransmitReceive(hspi2, &opcode_1, &reg_leido, 1, HAL_MAX_DELAY);  
    ;  
    HAL_SPI_TransmitReceive(hspi2, &opcode_2, &reg_leido, 1, HAL_MAX_DELAY);  
    ;  
    HAL_SPI_TransmitReceive(hspi2, &reg_LED_OFF, &reg_leido, 1,  
    HAL_MAX_DELAY); //LED OFF  
    HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_SET);  
  
    HAL_Delay(500);  
40 }  
  
void adc_send_command(uint8_t cmd, SPI_HandleTypeDef *SPI)  
{  
    uint8_t comando_temp = cmd;  
45    uint8_t zero = 0x00;  
    //PIN_MASTER_CS:  
    HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_RESET);  
    HAL_SPI_TransmitReceive(SPI, &comando_temp, &zero, 1, 100); //SPI.  
    transfer(cmd);  
    HAL_Delay(1);  
50    HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_SET);  
}
```



```
105 //      HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_SET) ;
//      while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==
//          GPIO_PIN_RESET) {}
}

void update_bias_ref(uint8_t data[], SPI_HandleTypeDef *SPI)
{
110     static uint8_t last_loff_statp = 0xFF;
     static uint8_t last_loff_statn = 0xFF;
     static unsigned samples_since_last_bias_change = 0;
     const unsigned min_samples_between_bias_changes = 100;

115     uint8_t loff_statp = frame_loff_statp(data);
     uint8_t leads_on_p = ~loff_statp;
     uint8_t loff_statn = frame_loff_statn(data);
     uint8_t leads_on_n = ~loff_statn;

120     bool shared_negative_electrode = true;

     if (shared_negative_electrode) {
         loff_statn |= 0x01; // count only the single shared electrode
125     }
     // if the lead-off status has changed...
     if (samples_since_last_bias_change >= min_samples_between_bias_changes
         && (last_loff_statp != loff_statp
         || last_loff_statn != loff_statn)) {

130         // Send SDATAC Command (Stop Read Data Continuously mode)
         // TODO: starting and stopping the data collection like this will
         // create a glitch in all channels of the recording whenever the
         // leadoff status of any channel changes. This could be fixed by
135         // capturing all data in single-shot mode, triggered by an interrupt.
         adc_send_command(SDATAC, SPI);

         // Use only the leads that are connected to drive the bias electrode.
         adc_wreg(RLD_SENSP, leads_on_p, SPI);
140         adc_wreg(RLD_SENSN, leads_on_n, SPI);

         // Put the Device Back in Read DATA Continuous Mode
         adc_send_command(RDATAC, SPI);

145         last_loff_statp = loff_statp;
         last_loff_statn = loff_statn;
         samples_since_last_bias_change = 0;
     } else {
         ++samples_since_last_bias_change;
150     }
}

//#ifdef __cplusplus
155     uint8_t frame_loff_statp(uint8_t data[])
     {
         return ((data[0] << 4) | (data[1] >> 4));
     }
}
```

```

    };
160   uint8_t frame_loff_statn(uint8_t data[])
    {
        return ((data[1] << 4) | (data[2] >> 4));
    };
    uint8_t frame_loff_statp_i(uint8_t data[], int i)
    {
        return ((frame_loff_statp(data) >> i) & 1);
    };
165   uint8_t frame_loff_statn_i(uint8_t data[], int i)
    {
        return ((frame_loff_statn(data) >> i) & 1);
    };
170   };
//#endif
//};

float32_t byte2float (uint8_t data_23_16, uint8_t data_15_8, uint8_t
175   data_7_0, uint8_t ganancia)
{
    float32_t value = 0;

    union miDato{
        struct
        {
            uint8_t b[4];           // Array de bytes de tamaño igual al tamaño de la
180   primera variable: int = 2 bytes, float = 4 bytes
            } split;
            long dato;
        } long_data;
185   long_data.dato = 0;

    if (data_23_16>=0x80)
    {
190       long_data.split.b[2] = ~data_23_16;
       long_data.split.b[1] = ~data_15_8;
       long_data.split.b[0] = ~data_7_0;
       value = long_data.dato;
       value = -value;
195   }
    else
    {
200       long_data.split.b[2] = data_23_16;
       long_data.split.b[1] = data_15_8;
       long_data.split.b[0] = data_7_0;
       value = long_data.dato;
    }

    value = value*4.5f/(8388607.0f*ganancia);
205   return value;
}

210 void configADS(uint8_t config[], uint8_t config_channel[] ,

```

```
SPI_HandleTypeDef *SPI, UART_HandleTypeDef *huart4)
{
    int i;
    bool continuo = false;

215    HAL_GPIO_WritePin(A_RESET_N_GPIO_Port, A_RESET_N_Pin, GPIO_PIN_RESET);
    HAL_Delay(100);
    HAL_GPIO_WritePin(A_RESET_N_GPIO_Port, A_RESET_N_Pin, GPIO_PIN_SET);

220    for (i = 0; i < 8; ++i) {
        HAL_Delay(50);
    }

    // Send SDATAC Command (Stop Read Data Continuously mode)
    adc_send_command(SDATAC, SPI);

225    // Power up the internal reference and wait for it to settle
    adc_wreg(CONFIG3, config[3-1], SPI);
    //adc_wreg(CONFIG3, PD_REFBUF | CONFIG3_reserved | BIASREF_INT |
    PD_BIAS, SPI); // Default mode
    //adc_wreg(CONFIG3, SPI); // 0xEC = 1110 1100

230    HAL_Delay(150);

    adc_wreg(CONFIG4, config[4-1], SPI);
    //adc_wreg(CONFIG4, PD_LOFF_COMP, SPI); // Default mode
235    //adc_wreg(CONFIG4, 0x02, SPI); // 0x02 = 0000 0010

    adc_wreg(LOFF, COMP_TH_80 | ILEAD_OFF_12nA, SPI);
    adc_wreg(LOFF_SENSP, 0xFF, SPI);
    adc_wreg(LOFF_SENSN, 0x01, SPI);

240    // Use lead-off sensing in all channels (but only drive one of the
    // negative leads if all of them are connected to one electrode)

    adc_wreg(CONFIG1, config[1-1], SPI); // 250 SPS
245    //adc_wreg(CONFIG1, CONFIG1_reserved | DR_250_SPS, SPI); // 250 SPS -
    Default mode - 0x96

    adc_wreg(CONFIG2, config[2-1], SPI);
    //adc_wreg(CONFIG2, CONFIG2_reserved | INT_CAL | CAL_FREQ_SLOW, SPI);
    // TEST_AMP | TEST_FREQ0); // generate internal test signals - Default
    mode
    //adc_wreg(CONFIG2, 0xD0, SPI); //D0 = 1101 0000

250    // If we want to share a single negative electrode, tie the negative
    // inputs together using the BIAS_IN line.
    //uint8_t mux = RLD_DRN;

    // connect the negative channel to the (shared) BIAS_IN line
    // Set the first LIVE_CHANNELS_NUM channels to input signal
255    for (i = 1; i <= LIVE_CHANNELS_NUM; ++i) {
        //adc_wreg(CHnSET + i, mux | GAIN_12X);
        //adc_wreg(CHnSET + i, ELECTRODE_INPUT | GAIN_12X, SPI);
        adc_wreg(CHnSET + i, config_channel[i-1], SPI);
```

```

    }
    // Set all remaining channels to shorted inputs
    for ( ; i <= 8; ++i) {
        adc_wreg(CHnSET + i, SHORTED | PDn, SPI);
265    }

    HAL_Delay(3 * 1000);

    HAL_GPIO_WritePin(A_START_GPIO_Port, A_START_Pin, GPIO_PIN_SET);
270    //adc_send_command(START, SPI);
    if (continuo == true)
    {
        adc_send_command(RDATAC, SPI);
275    } else
    {
        adc_send_command(SDATAC, SPI);
    }

280}

void acquire_single_data (uint8_t data[], SPI_HandleTypeDef *SPI,
    UART_HandleTypeDef *huart4)
{
// IPIN_MASTER_CS
285    adc_send_command(RDATAC, SPI);

    while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==
GPIO_PIN_SET) {}

290    HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_RESET);
    read_data_frame(data, SPI);

    while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==
GPIO_PIN_RESET) {}

295    HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_SET);
    update_bias_ref(data, SPI);
}

300

void acquire_array_data (uint8_t data[], float32_t channel_1[], float32_t
    channel_2[], float32_t channel_3[], float32_t channel_4[], float32_t
    channel_5[], float32_t channel_6[], float32_t channel_7[], float32_t
    channel_8[], uint8_t gain[], SPI_HandleTypeDef *SPI, UART_HandleTypeDef
    *huart4)
{
    int debug = 255;
305    // read LENGTH_SAMPLES samples

    //adc_wreg(GPIO, 0x2C, SPI);           // Led 1 on, led 2 off
}

```

```
adc_send_command(RDATAC, SPI);  
310  
while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==  
GPIO_PIN_SET){}  
  
HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_RESET);  
  
315  
read_data_frame(data, SPI);  
  
channel_1[0] = byte2float(data[1*3], data[1*3 +1], data[1*3 + 2], gain  
[0]);  
channel_2[0] = byte2float(data[2*3], data[2*3 +1], data[2*3 + 2], gain  
[1]);  
channel_3[0] = byte2float(data[3*3], data[3*3 +1], data[3*3 + 2], gain  
[2]);  
320  
channel_4[0] = byte2float(data[4*3], data[4*3 +1], data[4*3 + 2], gain  
[3]);  
channel_5[0] = byte2float(data[5*3], data[5*3 +1], data[5*3 + 2], gain  
[4]);  
channel_6[0] = byte2float(data[6*3], data[6*3 +1], data[6*3 + 2], gain  
[5]);  
channel_7[0] = byte2float(data[7*3], data[7*3 +1], data[7*3 + 2], gain  
[6]);  
channel_8[0] = byte2float(data[8*3], data[8*3 +1], data[8*3 + 2], gain  
[7]);  
325  
while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==  
GPIO_PIN_RESET){}  
  
for (int i = 1; i<LENGTH_SAMPLES; i++)  
{  
330  
    while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==  
GPIO_PIN_SET){}  
  
    read_data_frame(data, SPI);  
  
    channel_1[i] = byte2float(data[1*3], data[1*3 +1], data[1*3 + 2],  
gain[0]);  
335  
    channel_2[i] = byte2float(data[2*3], data[2*3 +1], data[2*3 + 2],  
gain[1]);  
    channel_3[i] = byte2float(data[3*3], data[3*3 +1], data[3*3 + 2],  
gain[2]);  
    channel_4[i] = byte2float(data[4*3], data[4*3 +1], data[4*3 + 2],  
gain[3]);  
    channel_5[i] = byte2float(data[5*3], data[5*3 +1], data[5*3 + 2],  
gain[4]);  
    channel_6[i] = byte2float(data[6*3], data[6*3 +1], data[6*3 + 2],  
gain[5]);  
340  
    channel_7[i] = byte2float(data[7*3], data[7*3 +1], data[7*3 + 2],  
gain[6]);  
    channel_8[i] = byte2float(data[8*3], data[8*3 +1], data[8*3 + 2],  
gain[7]);  
  
    while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==  
GPIO_PIN_RESET){}
```

```

345     }
346     HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_SET);

347     //adc_wreg(GPIO, 0x1C,SPI);           // Led 1 off , led 2 on
348     //Serial.println_N(debug, huart4);
349 }

350     void one_shot (uint8_t data[], SPI_HandleTypeDef *SPI,
351                   UART_HandleTypeDef *huart4)
352 {
353     uint8_t zero = 0x00;
354     uint8_t cmd = RDATA;
355     // IPIN_MASTER_CS

356     // while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==
357     // GPIO_PIN_SET) {}

358     HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_RESET);

359     HAL_SPI_TransmitReceive(SPI, &cmd, &zero, 1, 100);

360     read_data_frame(data, SPI);

361     // while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==
362     // GPIO_PIN_RESET) {}

363     HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_SET);

364     update_bias_ref(data, SPI);

365 }

366     void one_shot_array (uint8_t data[], float32_t channel_X[], uint8_t
367                          channel, uint8_t gain, SPI_HandleTypeDef *SPI, UART_HandleTypeDef *
368                          huart4)
369 {
370     uint8_t zero = 0x00;
371     uint8_t cmd = RDATA;
372     float32_t anterior = 0x00;
373     float32_t pre_anterior = 0x00;

374     // IPIN_MASTER_CS

375     // while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==
376     // GPIO_PIN_SET) {}
377     for (int i = 0; i < 2; i++)
378     {
379         HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_RESET);

380         HAL_SPI_TransmitReceive(SPI, &cmd, &zero, 1, 100);

381         // while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==
382         // GPIO_PIN_RESET) {}

383         read_data_frame(data, SPI);

```

```
channel_X[ i ] = byte2float( data[ channel*3 ] , data[ channel*3 +1] , data
[ channel*3 + 2] , gain );
// while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==
495 GPIO_PIN_RESET) {}
if (i==0){
    pre_anterior = channel_X[ i ] = byte2float( data[ channel*3 ] , data[ channel*3 +1] , data[ channel*3 + 2] , gain );
}
// else if (i==1){
//     anterior = channel_X[ i ] = byte2float( data[ channel*3 ] , data[ channel*3 +1] , data[ channel*3 + 2] );
400 //}
// HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_SET);

update_bias_ref( data , SPI );
}
405
for ( int i = 2; i<LENGTH_SAMPLES; )
{
    if (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==
        GPIO_PIN_RESET)
    {
        HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_RESET);
410
        HAL_SPI_TransmitReceive(SPI, &cmd, &zero, 1, 100);

        // while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==
            GPIO_PIN_RESET){}
415
        read_data_frame( data , SPI );

        channel_X[ i ] = byte2float( data[ channel*3 ] , data[ channel*3 +1] , data
[ channel*3 + 2] , gain );
        // while (HAL_GPIO_ReadPin(A_DRDY_N_GPIO_Port, A_DRDY_N_Pin) ==
420 GPIO_PIN_RESET) {}

        HAL_GPIO_WritePin(A_CS0_N_GPIO_Port, A_CS0_N_Pin, GPIO_PIN_SET);

        update_bias_ref( data , SPI );
        if (channel_X[ i ] != anterior){
425
            if (channel_X[ i-1 ] != pre_anterior){
                pre_anterior = anterior;
                anterior = channel_X[ i ];
                i++;
}
430
        }
    }
}
435 uint8_t calcular_ganancia ( uint8_t config_channel )
{
    uint8_t ganancia = 12;
```

```

440    switch ( config_channel&(0x70) )
441    {
442        case (0x00):
443            ganancia = 1;
444            break;
445
446        case (0x10):
447            ganancia = 2;
448            break;
449
450        case (0x30):
451            ganancia = 4;
452            break;
453
454        case (0x40):
455            ganancia = 8;
456            break;
457
458        case (0x50):
459            ganancia = 12;
460            break;
461
462        case (0x60):
463            ganancia = 24;
464            break;
465
466        default:
467            ganancia = 24;
468    }
469    return ganancia;
470
471 /*****END OF FILE*****
```

Código E.3: STM32F4:ADS1299.c

```

0  /**
1   ****
2
3   * File Name      : ADS1299.hpp
4   * Description     : This file contains the common defines of the
5   *                   ADS1299
6
7   ****
8
9   */
10 /* Define to prevent recursive inclusion
11
12 #ifndef __ADS1299_H
```

```
#define __ADS1299_H
#ifndef __cplusplus
    extern "C" {
15 #endif

    #include "stm32f4xx_hal.h"
    #include "arm_math.h"

20    // system commands
#define WAKEUP      0x02
#define STANDBY     0x04
#define RESSET      0x06
#define START       0x08
25 #define STOP       0x0A

    // read commands
#define RDATAAC     0x10
#define SDATAC      0x11
30 #define RDATA      0x12

    // register commands
#define RREG         0x20
#define WREG         0x40
35

    // device settings
#define ID          0x00

40    // global settings
#define CONFIG1     0x01
#define CONFIG2     0x02
#define CONFIG3     0x03
#define LOFF        0x04
45

    // channel specific settings
#define CHnSET      0x04
#define CH1SET      CHnSET + 1
#define CH2SET      CHnSET + 2
50 #define CH3SET      CHnSET + 3
#define CH4SET      CHnSET + 4
#define CH5SET      CHnSET + 5
#define CH6SET      CHnSET + 6
#define CH7SET      CHnSET + 7
55 #define CH8SET      CHnSET + 8
#define RLD_SENSP   0x0D
#define RLD_SENSN   0x0E
#define LOFF_SENSP  0x0F
#define LOFF_SENSN  0x10
60 #define LOFF_FLIP   0x11

    // lead off status
#define LOFF_STATP  0x12
#define LOFF_STATN  0x13
65

    // other
```

```

#define GPIO 0x14
#define PACE 0x15
#define RESP 0x16
70 #define CONFIG4 0x17
#define WCT1 0x18
#define WCT2 0x19

75 // enum CONFIG1_bits {
#define CONFIG1_reserved 0x90
#define DAISY_EN 0x40
#define CLK_EN 0x20

80 #define DR_250_SPS 0x06
#define DR_500_SPS 0x05
#define DR_1_kSPS 0x04
#define DR_2_kSPS 0x03
#define DR_4_kSPS 0x02
85 #define DR_8_kSPS 0x01
#define DR_16_kSPS 0x00

    // enum CONFIG2_bits {
#define CONFIG2_reserved 0xC0
90 #define INT_CAL 0x10
#define CAL_AMP 0x04
#define CAL_FREQ_SLOW 0x00
#define CAL_FREQ_FAST 0x01
#define CAL_FREQ_DC 0x03
95

    // enum CONFIG3_bits {
#define PD_REFBUF 0x80
#define CONFIG3_reserved 0x60
100 #define BIAS_MEAS 0x10
#define BIASREF_INT 0x08
#define PD_BIAS 0x04
#define BIAS_LOFF_SENS 0x02
#define BIAS_STAT 0x01
105

    // enum LOFF_bits {
#define COMP_TH2 0x80
#define COMP_TH1 0x40
110 #define COMP_TH0 0x20
#define VLEAD_OFF_EN 0x10
#define ILEAD_OFF1 0x08
#define ILEAD_OFF0 0x04
#define FLEAD_OFF1 0x02
115 #define FLEAD_OFF0 0x01

#define LOFF_const 0x00

    #define COMP_TH_95 0x00
120 #define COMP_TH_92_5 COMP_TH0
#define COMP_TH_90 COMP_TH1

```

```
#define COMP_TH_87_5 (COMP_TH1 | COMP_TH0)
#define COMP_TH_85 COMP_TH2
#define COMP_TH_80 (COMP_TH2 | COMP_TH0)
125 #define COMP_TH_75 (COMP_TH2 | COMP_TH1)
#define COMP_TH_70 (COMP_TH2 | COMP_TH1 | COMP_TH0)

#define ILEAD_OFF_6nA 0x00
#define ILEAD_OFF_12nA ILEAD_OFF0
130 #define ILEAD_OFF_18nA ILEAD_OFF1
#define ILEAD_OFF_24nA (ILEAD_OFF1 | ILEAD_OFF0)

#define FLEAD_OFF_AC FLEAD_OFF0
#define FLEAD_OFF_DC (FLEAD_OFF1 | FLEAD_OFF0)
135

    // enum CHnSET_bits {
#define PDn 0x80
#define PD_n 0x80
140 #define GAINn2 0x40
#define GAINn1 0x20
#define GAINn0 0x10
#define SRB2 0x08 // actually ADS1299 specific
#define MUXn2 0x04
145 #define MUXn1 0x02
#define MUXn0 0x01

#define CHnSET_const 0x00

150 #define GAIN_1X 0x00
#define GAIN_2X 0x10
#define GAIN_4X 0x20
#define GAIN_6X 0x30
#define GAIN_8X 0x40
155 #define GAIN_12X 0x50
#define GAIN_24X 0x60

#define ELECTRODE_INPUT 0x00
#define SHORTED MUXn0
160 #define RLD_INPUT MUXn1
#define MVDD (MUXn1 | MUXn0)
#define TEMP MUXn2
#define TEST_SIGNAL (MUXn2 | MUXn0)
#define RLD_DRP (MUXn2 | MUXn1)
165 #define RLD_DRN (MUXn2 | MUXn1 | MUXn0)

    // enum CH1SET_bits {
#define PD_1 0x80
170 #define GAIN12 0x40
#define GAIN11 0x20
#define GAIN10 0x10
#define MUX12 0x04
#define MUX11 0x02
175 #define MUX10 0x01
```

```

#define CH1SET_const 0x00

180 // enum CH2SET_bits {
#define PD_2 0x80
#define GAIN22 0x40
#define GAIN21 0x20
#define GAIN20 0x10
185 #define MUX22 0x04
#define MUX21 0x02
#define MUX20 0x01

#define CH2SET_const 0x00
190

// enum CH3SET_bits {
#define PD_3 0x80
#define GAIN32 0x40
195 #define GAIN31 0x20
#define GAIN30 0x10
#define MUX32 0x04
#define MUX31 0x02
#define MUX30 0x01
200 #define CH3SET_const 0x00

// enum CH4SET_bits {
205 #define PD_4 0x80
#define GAIN42 0x40
#define GAIN41 0x20
#define GAIN40 0x10
#define MUX42 0x04
210 #define MUX41 0x02
#define MUX40 0x01

#define CH4SET_const 0x00
215

// enum CH5SET_bits {
#define PD_5 0x80
#define GAIN52 0x40
#define GAIN51 0x20
220 #define GAIN50 0x10
#define MUX52 0x04
#define MUX51 0x02
#define MUX50 0x01
225 #define CH5SET_const 0x00

// enum CH6SET_bits {
230 #define PD_6 0x80
#define GAIN62 0x40
#define GAIN61 0x20

```

```
#define GAIN60 0x10
#define MUX62 0x04
#define MUX61 0x02
235 #define MUX60 0x01

#define CH6SET_const 0x00

240 // enum CH7SET_bits {
#define PD_7 0x80
#define GAIN72 0x40
#define GAIN71 0x20
#define GAIN70 0x10
245 #define MUX72 0x04
#define MUX71 0x02
#define MUX70 0x01

#define CH7SET_const 0x00
250

// enum CH8SET_bits {
#define PD_8 0x80
#define GAIN82 0x40
255 #define GAIN81 0x20
#define GAIN80 0x10
#define MUX82 0x04
#define MUX81 0x02
#define MUX80 0x01
260

#define CH8SET_const 0x00

265 // enum RLD_SENSP_bits {
#define RLD8P 0x80
#define RLD7P 0x40
#define RLD6P 0x20
#define RLD5P 0x10
#define RLD4P 0x08
270 #define RLD3P 0x04
#define RLD2P 0x02
#define RLD1P 0x01

#define RLD_SENSP_const 0x00
275

// enum RLD_SENSN_bits {
#define RLD8N 0x80
#define RLD7N 0x40
280 #define RLD6N 0x20
#define RLD5N 0x10
#define RLD4N 0x08
#define RLD3N 0x04
#define RLD2N 0x02
285 #define RLD1N 0x01
```

```

#define     RLD_SENSN_const 0x00

290 // enum LOFF_SENSP_bits {
#define     LOFF8P 0x80
#define     LOFF7P 0x40
#define     LOFF6P 0x20
#define     LOFF5P 0x10
295 #define     LOFF4P 0x08
#define     LOFF3P 0x04
#define     LOFF2P 0x02
#define     LOFF1P 0x01

300 #define     LOFF_SENSP_const 0x00

305 // enum LOFF_SESN_N_bits {
#define     LOFF8N 0x80
#define     LOFF7N 0x40
#define     LOFF6N 0x20
#define     LOFF5N 0x10
#define     LOFF4N 0x08
#define     LOFF3N 0x04
310 #define     LOFF2N 0x02
#define     LOFF1N 0x01

315 #define     LOFF_SESN_N_const 0x00

320 // enum LOFF_FLIP_bits {
#define     LOFF_FLIP8 0x80
#define     LOFF_FLIP7 0x40
#define     LOFF_FLIP6 0x20
#define     LOFF_FLIP5 0x10
#define     LOFF_FLIP4 0x08
#define     LOFF_FLIP3 0x04
#define     LOFF_FLIP2 0x02
#define     LOFF_FLIP1 0x01
325 #define     LOFF_FLIP_const 0x00

330 // enum LOFF_STATP_bits {
#define     IN8P_OFF 0x80
#define     IN7P_OFF 0x40
#define     IN6P_OFF 0x20
#define     IN5P_OFF 0x10
#define     IN4P_OFF 0x08
335 #define     IN3P_OFF 0x04
#define     IN2P_OFF 0x02
#define     IN1P_OFF 0x01

340 #define     LOFF_STATP_const 0x00

```

```
// enum LOFF_STATN_bits {
#define IN8N_OFF 0x80
#define IN7N_OFF 0x40
345 #define IN6N_OFF 0x20
#define IN5N_OFF 0x10
#define IN4N_OFF 0x08
#define IN3N_OFF 0x04
#define IN2N_OFF 0x02
350 #define IN1N_OFF 0x01

#define LOFF_STATN_const 0x00

355 // enum GPIO_bits {
#define GPIOD4 0x80
#define GPIOD3 0x40
#define GPIOD2 0x20
#define GPIOD1 0x10
360 #define GPIOC4 0x08
#define GPIOC3 0x04
#define GPIOC2 0x02
#define GPIOC1 0x01

365 #define GPIO_const 0x0F

// enum PACE_bits {
#define PACEE1 0x10
370 #define PACEE0 0x08
#define PACEO1 0x04
#define PACEO0 0x02
#define PD_PACE 0x01

375 #define PACE_const 0x00

#define PACEE_CHAN2 0x00
#define PACEE_CHAN4 PACEE0
#define PACEE_CHAN6 PACEE1
380 #define PACEE_CHAN8 (PACEE1 | PACEE0)

#define PACEO_CHAN1 0x00
#define PACEO_CHAN3 PACEE0
#define PACEO_CHAN5 PACEE1
385 #define PACEO_CHAN7 (PACEE1 | PACEE0)

// enum RESP_bits {
#define RESP_DEMOD_EN1 0x80
#define RESP_MOD_EN1 0x40
390 #define RESP_PH2 0x10
#define RESP_PH1 0x08
#define RESP_PH0 0x04
#define RESP_CTRL1 0x02
#define RESP_CTRL0 0x01
395 #define RESP_const 0x20
```

```

400 //define RESP_PH_22_5 0x00
#define RESP_PH_45 RESP_PH0
#define RESP_PH_67_5 RESP_PH1
#define RESP_PH_90 (RESP_PH1 | RESP_PH0)
#define RESP_PH_112_5 RESP_PH2
#define RESP_PH_135 (RESP_PH2 | RESP_PH0)
#define RESP_PH_157_5 (RESP_PH2 | RESP_PH1)

405 #define RESP_NONE 0x00
#define RESP_EXT RESP_CTRL0
#define RESP_INT_SIG_INT RESP_CTRL1
#define RESP_INT_SIG_EXT (RESP_CTRL1 | RESP_CTRL0)

410 // enum CONFIG4_bits {
#define SINGLE_SHOT 0x08
#define PD_LOFF_COMP 0x02

415 #define CONFIG4_reserved 0x00

#define RESP_FREQ_64k_Hz 0x00
#define RESP_FREQ_32k_Hz RESP_FREQ0
420 #define RESP_FREQ_16k_Hz RESP_FREQ1
#define RESP_FREQ_8k_Hz (RESP_FREQ1 | RESP_FREQ0)
#define RESP_FREQ_4k_Hz RESP_FREQ2
#define RESP_FREQ_2k_Hz (RESP_FREQ2 | RESP_FREQ0)
#define RESP_FREQ_1k_Hz (RESP_FREQ2 | RESP_FREQ1)
425 #define RESP_FREQ_500_Hz (RESP_FREQ2 | RESP_FREQ1 | RESP_FREQ0)

430 //enum WCT1_bits {
#define aVF_CH6 0x80
#define aVL_CH5 0x40
#define AVR_CH7 0x20
#define avR_CH4 0x10
#define PD_WCTA 0x08
#define WCTA2 0x04
435 #define WCTA1 0x02
#define WCTA0 0x01

#define WCT1_const 0x00

440 #define WCTA_CH1P 0x00
#define WCTA_CH1N WCTA0
#define WCTA_CH2P WCTA1
#define WCTA_CH2N (WCTA1 | WCTA0)
#define WCTA_CH3P WCTA2
445 #define WCTA_CH3N (WCTA2 | WCTA0)
#define WCTA_CH4P (WCTA2 | WCTA1)
#define WCTA_CH4N (WCTA2 | WCTA1 | WCTA0)

450 //enum WCT2_bits {
#define PD_WCTC 0x80

```

```
#define PD_WCTB 0x40
#define WCTB2 0x20
#define WCTB1 0x10
455 #define WCTB0 0x08
#define WCTC2 0x04
#define WCTC1 0x02
#define WCTC0 0x01

460 #define WCT2_const 0x00

#define WCTB_CH1P 0x00
#define WCTB_CH1N WCTB0
#define WCTB_CH2P WCTB1
465 #define WCTB_CH2N (WCTB1 | WCTB0)
#define WCTB_CH3P WCTB2
#define WCTB_CH3N (WCTB2 | WCTB0)
#define WCTB_CH4P (WCTB2 | WCTB1)
#define WCTB_CH4N (WCTB2 | WCTB1 | WCTB0)

470 #define WCTC_CH1P 0x00
#define WCTC_CH1N WCTC0
#define WCTC_CH2P WCTC1
#define WCTC_CH2N (WCTC1 | WCTC0)
475 #define WCTC_CH3P WCTC2
#define WCTC_CH3N (WCTC2 | WCTC0)
#define WCTC_CH4P (WCTC2 | WCTC1)
#define WCTC_CH4N (WCTC2 | WCTC1 | WCTC0)

480 #define size 27

    // actual size today is 64, but a few extra will not hurt
#define DATA_BUF_SIZE 80
485 // Tamaño del buffer de la señal a leer
#define DATA_LONG 256
#define CANAL 1

490 #ifndef LIVE_CHANNELS_NUM
#define LIVE_CHANNELS_NUM 8
#endif

495 void Blinky(SPI_HandleTypeDef *hspi2);
void adc_send_command(uint8_t cmd, SPI_HandleTypeDef *SPI);
uint8_t adc_rreg(uint8_t reg, SPI_HandleTypeDef *SPI);
void adc_wreg(uint8_t reg, uint8_t val, SPI_HandleTypeDef *SPI);
void read_data_frame(uint8_t data[], SPI_HandleTypeDef *SPI);
500 void update_bias_ref(uint8_t data[], SPI_HandleTypeDef *SPI);
long extrae_un_canal (uint8_t data[], uint8_t canal);
void format_data_frame(uint8_t data[], char *byte_buf);
void print_chip_id(SPI_HandleTypeDef *SPI, UART_HandleTypeDef *huart4);
float32_t byte2float (uint8_t data_23_16, uint8_t data_15_8, uint8_t
    data_7_0, uint8_t ganancia);
505 void configADS(uint8_t config[], uint8_t config_channel[],
```

```

    SPI_HandleTypeDef *SPI, UART_HandleTypeDef *huart4);
void acquire_single_data (uint8_t data[], SPI_HandleTypeDef *SPI,
    UART_HandleTypeDef *huart4);
void acquire_array_data (
    uint8_t data[],
    float32_t channel_1[],
    float32_t channel_2[],
    float32_t channel_3[],
    float32_t channel_4[],
    float32_t channel_5[],
    float32_t channel_6[],
    float32_t channel_7[],
    float32_t channel_8[],
510   uint8_t gain[], SPI_HandleTypeDef *SPI, UART_HandleTypeDef *huart4);
void one_shot (uint8_t data[], SPI_HandleTypeDef *SPI, UART_HandleTypeDef *
    huart4);
void one_shot_array (uint8_t data[], float32_t channel_X[], uint8_t channel,
    uint8_t gain, SPI_HandleTypeDef *SPI, UART_HandleTypeDef *huart4);
515   uint8_t calcular_ganancia (uint8_t config_channel);

    uint8_t frame_loff_statp(uint8_t data[]);
    uint8_t frame_loff_statn(uint8_t data[]);
    uint8_t frame_loff_statp_i(uint8_t data[], int i);
520   uint8_t frame_loff_statn_i(uint8_t data[], int i);

#ifdef __cplusplus
}
#endif // Cplusplus
525 #endif /* _ADS1299_H */
*****END OF FILE*****

```

Código E.4: STM32F4:ADS1299.h