Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Procesamiento de ondas cerebrales con microprocesador ARM para control de coche teledirigido

Autor: Daniel Calderón Martínez

Tutor: Vicente Baena Lecuyer

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías de Telecomunicación

Procesamiento de ondas cerebrales con microprocesador ARM para control de coche teledirigido

Autor:
Daniel Calderón Martínez

Tutor:
Vicente Baena Lecuyer
Profesor titular

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2016

Trabajo Fin de Grado: Procesamiento de ondas cerebrales con microprocesador ARM para control de coche teledirigido

Autor: Daniel Calderón Martínez

Tutor: Vicente Baena Lecuyer

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

A mis padres

Agradecimientos

Ya que este es el único espacio personal de entre todas estas páginas, me gustaría no dejar pasar la oportunidad de expresar en estas líneas mi agradecimiento. En primer lugar y más importante, he de dar las gracias con mayúsculas, a las personas sin las cuales hoy no estaría escribiendo estas líneas. Las personas que han estado a mi lado incondicionalmente durante estos años, más o menos duros, en esta carrera de fondo de la que veo ya el ansiado final. Esas personas que desde el minuto uno de mi vida, tomaron las decisiones oportunas para poder ofrecerme los recursos necesarios para estudiar y llegar hasta este nivel. Esas personas por supuesto son mis padres, Salvador y Josefa. No creo que haga falta añadir nada más. Simplemente Gracias.

Por otro lado, quiero mostrar también mi agradecimiento a mi tutor, D. Vicente Baena Lecuyer, quién me ha aportado las ideas y ayuda necesaria en todo momento. Gracias por aceptarme como alumno para realizar este trabajo y gracias por la gran disponibilidad que ha tenido hacia mi persona durante todo este tiempo.

No podría faltar tampoco mi agradecimiento a José García Doblado, por las librerías proporcionadas y su ayuda con respecto al microprocesador utilizado.

Daniel Calderón Martínez

Sevilla, 2016

Resumen

Este trabajo fin de grado trata sobre el procesamiento de las señales que produce el cerebro humano para utilizarlas y destinarlas a algún propósito. En este caso, dicho propósito es el control de un coche teledirigido de juguete. Mediante un dispositivo electrónico comercial, que realiza la función de BCI (*Brain Computer Interface)*, se captan las ondas cerebrales haciendo uso de una electroencefalografía (EEG) y son enviadas a un microprocesador ARM vía bluetooth. El casco o sensor de ondas, implementa un algoritmo que indica los niveles de atención y meditación. A través de estos niveles y con las propias ondas, se realiza el procesamiento en el microprocesador para darle la funcionalidad correspondiente. Mediante un nivel alto de atención, el coche irá hacia delante, mientras que con la relajación, el coche irá hacia atrás. A su vez se identifica el parpadeo en la señal reciba, observando los picos producidos, para dirigir las ruedas del coche hacia la izquierda o hacia la derecha.

Abstract

This final degree work deals with the processing of the signals produced by the human brain for use and destine them to some purpose. In this case, that purpose is to control a remote control toy car. Using a commercial electronic device, brain waves are captured using an electroencephalogram (EEG) and are sent to the ARM microprocessor via Bluetooth. The hull or wave sensor, implements an algorithm that indicates the levels of care and meditation. Through these levels and the waves themselves, processing is performed in the microprocessor to give the corresponding functionality. Through a high level of attention, the car go forward, while the relaxation, the car will go backward. At the same time the blinking signal received serve to direct the wheels of the car to the left or right.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

Acrónimos

BCI	Brain computer interface / Interfaz cerebro-ordenador
EEG	Electroencelografía
ECoG	Electrocorticografía
MEG	Magnetoencefalografía
FP	Punto Frontal Polar
O	Punto Occipital
Fz	Punto Frontal
Cz	Punto Central
Pz	Punto Parietal
T	Puntos Temporales
C	Puntos Centrales
RISC	Reduced Instruction Set Computer / Ordenador con conjunto de instrucciones reducidos
DSP	Digital Signal Processor / Procesador digital de señales
FPU	Float Point Unit / Unidad de punto flotante
NVIC	Nested Vectores Interrupt Controller / Controlador de vector de interrupciones anidadas
MPU	Memory Protection Unit / Unidad de protección de memoria
WIC	Wake-up Interrupt Controller / Controlador de despertador de interrupciones
AHB	Advance High-performance Bus / Bus avanzado de alto rendimiento
APB	Advanced Peripheral Bus / Bus de periférico avanzado
PPB	Private Peripheral Bus / Bus de periférico privado
SW-DP	Serial Wire Debug Port / Puerto de depuración Serial Wire
JTAG-DP	JTAG Debug Port / Puerto de depuración JTAG
SRAM	Static Random-Access Memory
RTC	Real-Time Clock / Reloj en tiempo real
PWM	Pulse-Width Modulation / Modulación por ancho de pulsos
USB-OTG	Universal Serial Bus - On the Go
PSRAM	Pseudostatic RAM
LCD	Liquid Crystal Display / Pantalla de cristal líquido
A/D	Convertidor analógico/digital
DMA	Direct Memory Access / Acceso directo a memoria
I2C	Inter-Integrated Circuit / Circuito Inter-integrado
USART	Universal Synchronous/Asynchronous Receiver/Transmitter

UART	Universal Asynchronous Receiver/Transmitter
SPI	Serial Peripheral Interface / Interfaz de periférico en serie
CAN	Controller Area Network / Red de Control de Área
SDIO	Secure Digital Input Output / Entrada-Salida Digital Segura
CRC	Cyclic Redundancy Check / Comprobación de redundancia cíclica
DAC	Digital Analog Converter / Convertidor digital/analógico

1 INTRODUCCIÓN

En mi comienzo está mi final.

- T.S.Eliot -

1.1 Antecedentes

Hoy día la tecnología sigue creciendo a pasos agigantados, tanto es así, que lo que hace años atrás era ficción, hoy día es realidad. Por ejemplo coches que se conducen sin piloto. Hago referencia a esto ya que este proyecto trata sobre "leer la mente para mover un coche de juguete". Obviamente esto no es tal que así, pero podría tener una lectura similar. Se está llegando a este punto, al punto de poder usar nuestro órgano más potente para el beneficio propio. Aunque la finalidad o aplicación de este trabajo es la comentada anteriormente, la misma se podría extrapolar a algo que hoy día está avanzando mucho como es la biomedicina. ¿Podrán las personas sin movilidad usar su mente en favor propio? Este campo es muy amplio y se está investigando y trabajando en él duramente. Este trabajo consiste en aprovechar parte de lo ya estudiado y aplicarlo a través de un microprocesador.

1.2 Objetivos y alcance

El fin de este trabajo consiste en aplicar conocimientos técnicos adquiridos durante estos años de estudio a la vez de adquirir otros nuevos, y utilizarlos en pro de mover un coche teledirigido haciendo uso de la tecnología de los microprocesadores y un sensor de ondas cerebrales comercial.

1.3 Organización del documento

Tras este capítulo introductorio, en un primer lugar se hará una pequeña recopilación de información acerca del cerebro humano, su estructura y de la propia electroencefalografía (EEG).

Posteriormente, en un nuevo capítulo se tratará la cuestión relacionada con la interfaz cerebro-ordenador (BCI), realizando primeramente una introducción acerca de esta tecnología para a continuación desarrollarla más profundamente. A su vez en este mismo punto se describirá el dispositivo utilizado en este proyecto como BCI.

Tras el BCI, se tratarán en otro capítulo todos los aspectos relacionados con el microprocesador ARM, explicando el modelo y la placa de evaluación utilizada, así como el entorno de desarrollo en el que se ha realizado este proyecto.

En un nuevo apartado, se expondrá la aplicación desarrollada en este trabajo. Explicando el funcionamiento mismo, así como el detalle a nivel electrónico del coche teledirigido empleado.

Por último, se darán las conclusiones y se hará una pequeña reseña acerca del futuro de las tecnologías de las BCI y hacia dónde van encaminadas.

2 EL CEREBRO HUMANO

2.1 Introducción

El cerebro humano está formado por una precisa y compleja red de neuronas cuya tarea es la de transmitir la información mediante actividad electroquímica, haciendo que se creen campos eléctricos. Dichos campos eléctricos pueden ser evaluados. A esta medición del campo eléctrico generado por el cerebro se denomina electroencefalografía (EEG) y al campo, ondas cerebrales. Según el estado de la persona, el patrón de las ondas será de una forma u otra. Estos ritmos cerebrales pueden ser capturados y procesados para un fin. Esta es la tarea de las interfaces cerebro-ordenador (BCI).

2.2 Estructura del cerebro

El encéfalo, que está contenido en el cráneo, es la parte más grande del sistema nervioso. Se divide en tres partes: cerebelo, tallo cerebral y cerebro.

El cerebelo tiene la tarea de mantener el equilibrio del cuerpo y coordinar sus movimientos.

El tallo cerebral es el encargado de controlar los ritmos cardíacos y respiratorios, además de ser el centro de los reflejos.

Por último, el cerebro es el encargado de dar sentido a nuestras vidas. Es el centro del sistema nervioso y por ello es el encargado de controlarnos a todos los niveles. Nuestra mente y conciencia dependen de él.

Diferentes regiones forman el cerebro [1], pero quizás las más importantes o que más nos interesan sean los cuatro lóbulos (figura 2-1):

- Lóbulo frontal: este lóbulo solo lo poseen de forma desarrollada los animales más complejos [2]. Es el encargado del habla o expresión oral.
- Lóbulo parietal: es el encargado de procesar la información sensorial como el tacto, el calor o el frío y también se encarga de coordinar el equilibrio [3].
- Lóbulo occipital: se encarga de la percepción visual, es decir, el responsable de procesar las imágenes [4].
- Lóbulo temporal: se relaciona con la memoria y a su vez se encarga de regular las emociones y motivaciones [5].

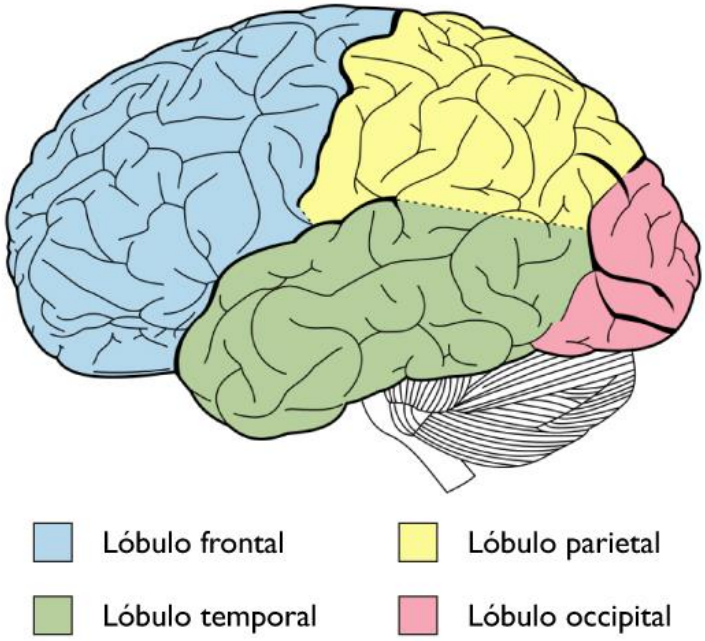

Figura 2-1 Visión lateral de los lóbulos cerebrales [1].

2.3 Impulsos eléctricos

Los impulsos eléctricos o potenciales de acción, son ondas de descargas eléctricas que recorren la membrana celular produciendo cambios en la carga eléctrica. Sirven para llevar información entre unos tejidos y otros.

Los impulsos eléctricos producen ritmos que son conocidos como ondas cerebrales.

Las células encargadas de la recepción de estímulos y la conducción de los impulsos eléctricos son las neuronas, cuya interconexión se denomina sinapsis. Éstas son las responsables de interconectar los componentes del sistema nervioso: motor, integrador y sensitivo. Tienen la capacidad de comunicarse con precisión con otras células, musculares, nerviosas o glandulares. Los impulsos nerviosos viajan por la neurona desde la dendrita, pasando por toda la neurona hasta llegar a otra neurona, algún musculo o alguna glándula.

Las partes más importantes de una neurona son:

- Soma o cuerpo celular: es el propio cuerpo de la neurona.
- Núcleo: es la parte más grande e importante, suele ocupar una posición central.
- Dendritas: son prolongaciones del cuerpo celular. Encargadas de recibir los impulsos y pasarlos al soma.
- Axón: es una prolongación cuya función es la de transportar la información de la neurona a su destino.

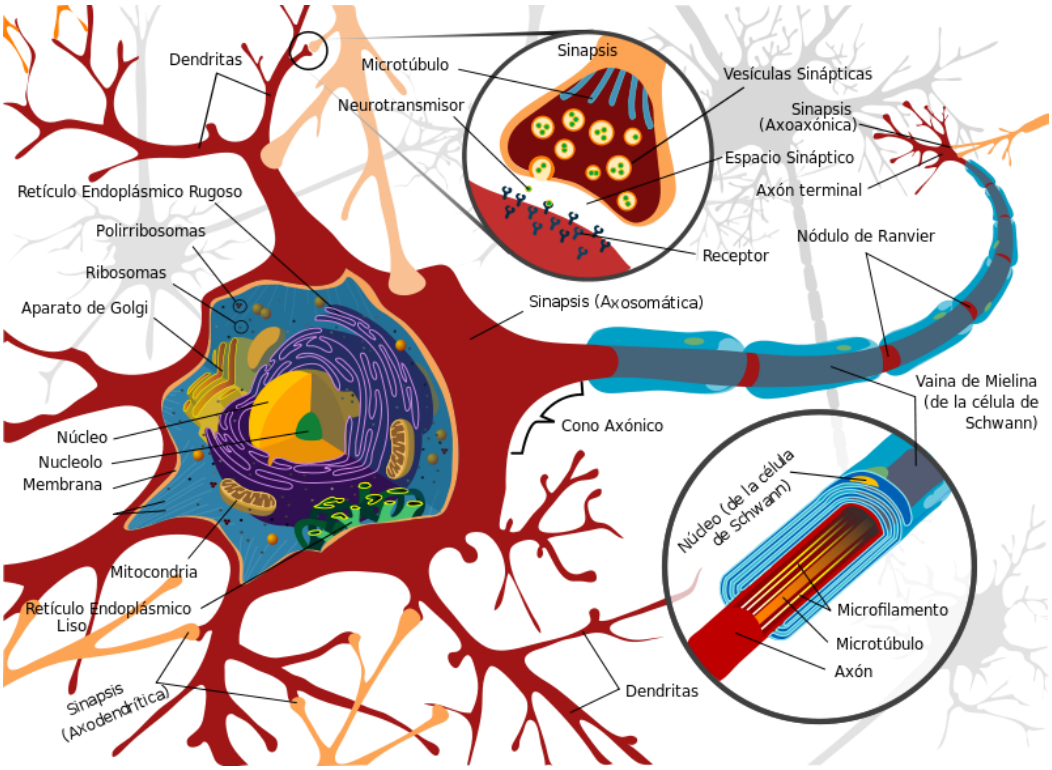

Figura 2-2 Partes de una neurona [12].

La propagación de las neuronas se debe a un cambio de potencial y las ondas que transmiten son eléctricas. La diferencia de potencial entre la parte interna y externa de la membrana celular es por lo general de -70mV [6]. Cuando se despolariza el potencial de la membrana, la célula genera el potencial de acción.

En resumen, un potencial de acción es un cambio muy rápido en la polaridad de la membrana celular en un corto periodo de tiempo.

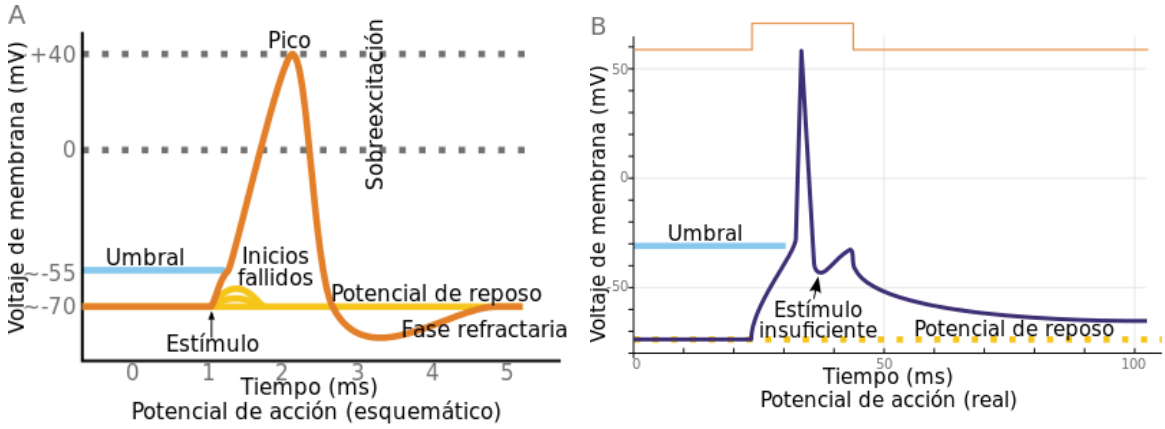

Figura 2-3 A. Potencial de acción ideal. **B.** Potencial de acción real [7].

2.4 Electroencefalografía

2.4.1 Introducción

La electroencefalografía (EEG) es una técnica mediante la cual se observa y registra la actividad eléctrica generada en el cerebro. Esta medida se obtiene a través de electrodos colocados en posiciones concretas de la cabeza. Son electrodos muy sensibles, ya que la escala de la señal eléctrica va desde 10 milivoltios sobre el córtex hasta los 100 microvoltios en la superficie del cuero cabelludo [9].

2.4.2 Captación de las ondas cerebrales

Las señales EEG se muestrean a una frecuencia de 100 Hz o mayor. Los equipos modernos pueden muestrear hasta una frecuencia de 500 Hz [10]. La captación de las ondas cerebrales puede realizarse de diferentes maneras, según la posición de los electrodos. Estos pueden ser de tres tipos: superficiales, basales y quirúrgicos [9]. Además según el tipo de electrodo usado la captación de las ondas se denomina de una forma u otra.

Si los electrodos son superficiales o basales, se denomina Electroencefalograma (EEG). Cuando los que se usan son electrodos quirúrgicos, a la captación se le llama Electrocorticograma (ECoG). Finalmente si se emplean electrodos quirúrgicos introducidos más profundamente, se designa el término Estereo Electroencefalograma (E-EEG).

Dentro de la clasificación de los electrodos se realizan otras subdivisiones. Los electrodos superficiales, a su vez, se dividen en:

- Superficiales de contacto.
- En casco de malla.
- Adheridos.
- De aguja.
- Quirúrgicos.

2.4.3 Posición de los electrodos

Hay diferentes sistemas de posicionamiento de electrodos (Aird, Cohn, Illinois, etc.), sin embargo el más utilizado es el “10/20” [8], que consiste en poner entre 16 y 20 electrodos en una localización específica. El procedimiento de este sistema se muestra en la figura 2-3. Actualmente se usan gorras con los electrodos incorporados para que sea más rápida la colocación de estos. Este sería el sistema más eficaz de realizar un EEG, ya que hay multitud de electrodos, y por lo tanto multitud de puntos para medir las señales eléctricas. Sin embargo hay dispositivos más sencillos, con menos electrodos, que pueden ser útiles dependiendo del fin del EEG. En este trabajo, se va a trabajar con un sensor que posee solo un electrodo. Pero esto se comentará más adelante.

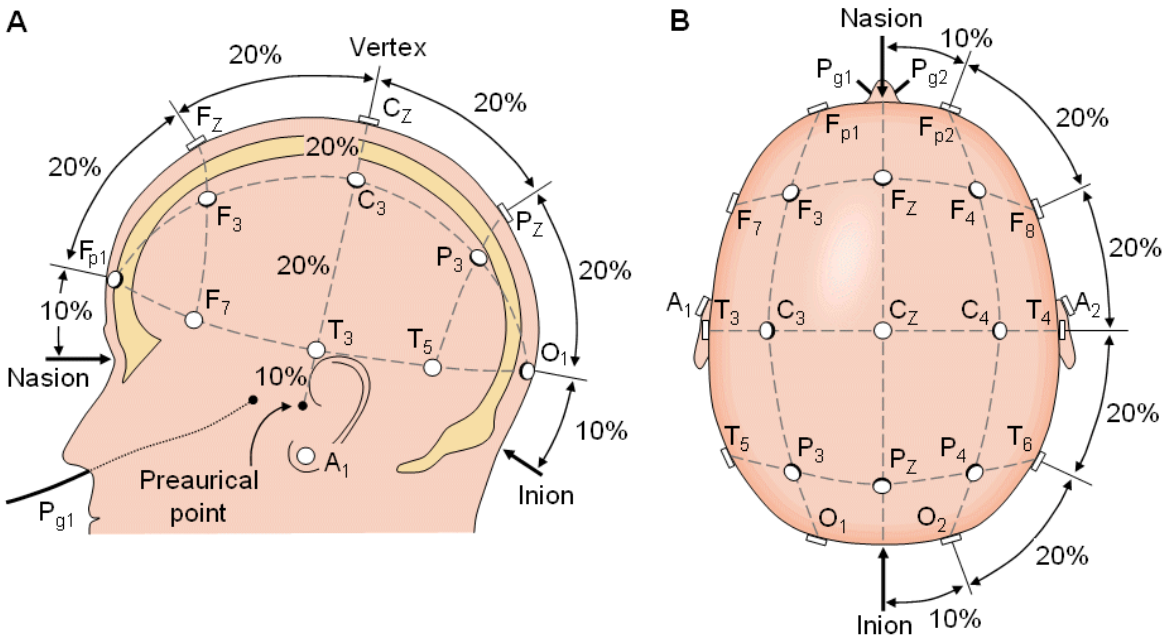

Figura 2-4 Posicionamiento de electrodos según el sistema 10/20 [11].

2.4.4 Ondas recibidas vía EEG

Las ondas cerebrales recibidas al realizar el electroencefalograma, también denominadas ritmos, se clasifican según la banda de frecuencia a la que pertenecen:

Nombre	Rango de Frecuencias (Hz)
Delta	0.5 – 3.5
Theta	3.5 – 7.5
Alfa	7.5 – 12.5
Beta	12.5 – 30
Gamma	30 – 60

Tabla 2-1 Bandas de frecuencia de los ritmos cerebrales.

- Ondas Delta [0.5 – 3.5 Hz]: características de estados profundos de sueño, en la niñez y en enfermedades cerebrales graves.

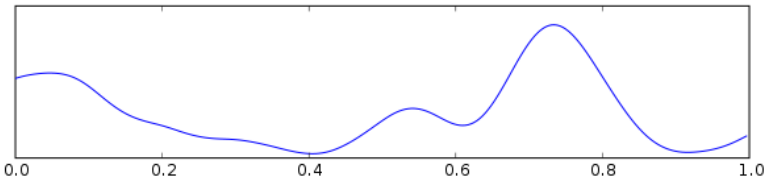

Figura 2-5 Ritmo Delta.

- Ondas Theta [3.5 – 7.5 Hz]: aparecen durante el sueño también. En niños son normales, pero en adultos son muestra de desórdenes mentales, stress o problemas como la epilepsia.

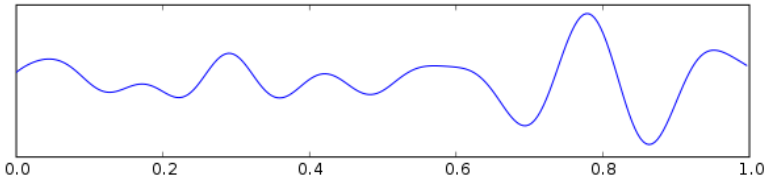

Figura 2-6 Ritmo Theta.

- Ondas Alfa [7.5 – 12.5 Hz]: son características en personas despiertas, con los ojos cerrados y sin actividad mental.

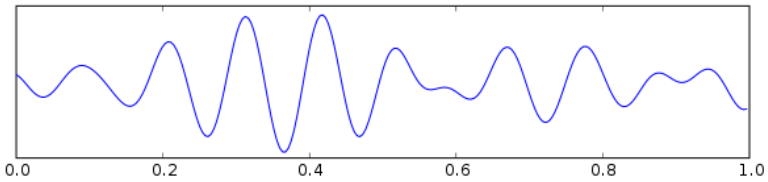

Figura 2-7 Ritmo Alfa.

- Ondas Beta [12.5 – 30 Hz]: se encuentran definidas en las regiones parietal y frontal. Tienen menos amplitud que las ondas alfa. Se producen cuando la persona está bajo tensión, expectante o realizando cálculos mentales.

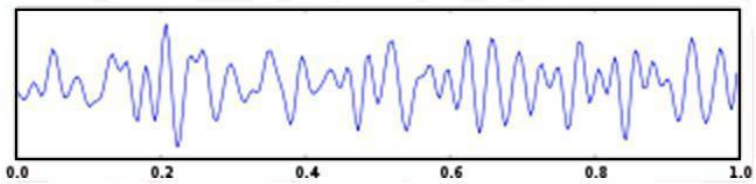

Figura 2-8 Ritmo Beta.

- Ondas Gamma [30 – 60 Hz]: aparecen en estados de atención o concentración.

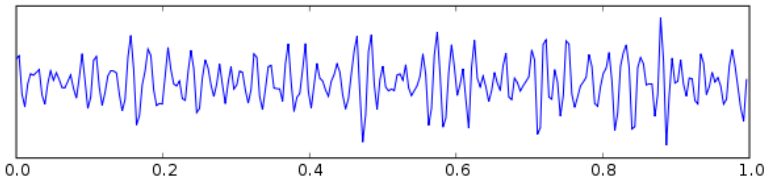

Figura 2-9 Ritmo Gamma.

En este proyecto, los ritmos que más nos interesan son los Theta, en el caso de querer medir la relajación y que el coche vaya hacia atrás. Y el ritmo Gamma, que sería señal de un nivel de atención elevado; haciendo que el coche fuera hacia delante.

3 INTERFAZ CEREBRO-ORDENADOR

3.1 Introducción

Hoy día el poder interactuar con las máquinas a través de nuestros pensamientos es algo que se está estudiando de manera muy intensa. La idea o uso de esta tecnología no es algo que esté muy alejado del alcance de cualquiera. Las acciones a realizar con la mente, pueden ir desde tareas sencillas como encender una bombilla, a ejercicios complicados como el controlar una silla de ruedas. Aunque la idea no parece muy disparatada, la cuestión tecnológica no es evidente. Esto es debido a que confluyen diferentes tipos de disciplinas, como la neurociencia, la biomédica y la computación.

Esta tecnología evoluciona a gran velocidad, no solo en el campo de la medicina sino también en el del ocio. Podemos ver juguetes que ya están comercializados, que trabajan con la BCI. Lo que es seguro, es que en el futuro será algo que estará más que implantado.

Pero, ¿qué es realmente una BCI? Las interfaces cerebro-ordenador proporcionan a los usuarios comunicación y control sobre cierto dispositivo, dependiendo únicamente de la actividad cerebral, sin utilizar ningún músculo. La BCI se basa en tres puntos: sensor, procesamiento de la señal y la aplicación. Cualquier uso de las BCI's están relacionadas con alguno de esos puntos o el conjunto de ellos. En este caso, nos centraremos más en la aplicación.

Las señales cerebrales a procesar, pueden provenir de diferentes técnicas: EEG, ECoG o MEG. El método más usado, ya que no es invasivo y es más sencillo, es el EEG.

3.2 Modelo funcional genérico

El principio de funcionamiento es el de captar la actividad cerebral, procesarla para obtener las características que interesan y después utilizarlas para interaccionar con el dispositivo o aplicación elegidos. Cabe destacar, que la BCI, se basa meramente en las señales cerebrales, sin incluir la actividad eléctrica producida por los músculos, denominados artefactos.

Los pasos a seguir que describen el funcionamiento de una BCI, son los siguientes [13]:

1. Adquisición de la señal: en este bloque se adquiere la señal con el sensor utilizado en cada caso, se amplifica y se realiza la conversión Analógico/Digital.

2. Procesamiento: hay que distinguir 3 etapas en este punto:

- Cancelación de artefactos. Aquí es donde se eliminan los artefactos, que son los ruidos provenientes de actividad eléctrica relacionada con los músculos y que distorsionan la señal.

- Obtención de las características. Se traduce la señal de entrada en un vector de características en relación al fenómeno neurológico asociado a la señal.

- Decodificación. Finalmente las características obtenidas se traducen en las señales de control que se usaran en el dispositivo a controlar.

3. Aplicación: este bloque recibe del anterior las señales de control y realiza las accionas configuradas en función de dichas señales.

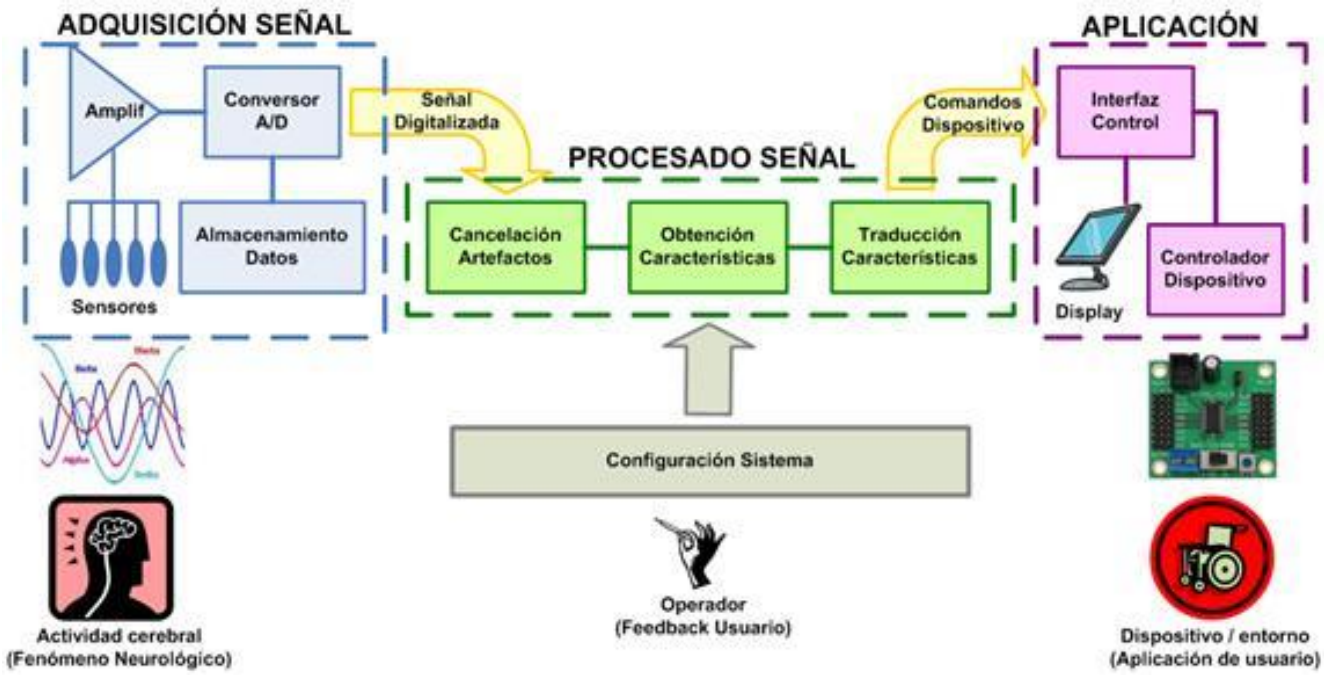

Figura 3-1 Modelo funcional de una BCI [14].

3.3 Mindwave Mobile

3.3.1 Introducción

Anteriormente se ha hablado de la interfaz cerebro-ordenador, realizando una introducción y explicando el proceso de funcionamiento de dicha tecnología. En este apartado se hace alusión al dispositivo comercial utilizado en este proyecto. Se definirán las características más importantes y se explicará el funcionamiento del mismo.

Figura 3-2 Mindwave Mobile [15].

3.3.2 Características

El dispositivo MindWave es una diadema que recoge la actividad cerebral y divide las señales según la frecuencia. Esta medida se realiza situando el sensor de contacto en el FP1 y el nodo de referencia con el clic, que se puede observar en la figura, en la oreja. La mayoría de las aplicaciones a realizar con este dispositivo son de uso práctico y no muy complejas, ya que solo tenemos un sensor, por lo tanto, un solo canal a medir.

El Mindwave es capaz de medir, principalmente, los estados de relajación y atención. De hecho la compañía, *Neurosky*, ha implementado un algoritmo llamado *eSense*, que mide los niveles de atención y relajación en una escala de 0 a 100. Todo esto a través de los ritmos cerebrales: alfa, beta, gamma… También es capaz de captar el parpadeo, herramienta que utilizaremos en la aplicación de este proyecto.

El paquete Mindwave Mobile, incorpora un CD de instalación, con una aplicación para el ordenador. De forma que cuando te conectas al ordenador con el sensor, pueden visualizar las ondas cerebrales clasificadas en frecuencia. A su vez, incluye dos juegos mentales. Uno consiste en hacer explotar un barril con un nivel alto de atención y el otro sirve para hacer levitar una bola mediante la relajación. Dichos juegos sirven para entrenar la mente de cara a la aplicación a utilizar. También se incluye en el CD un juego de velocidad de cálculos matemáticos del que se extraen ciertas medidas de las ondas cerebrales.

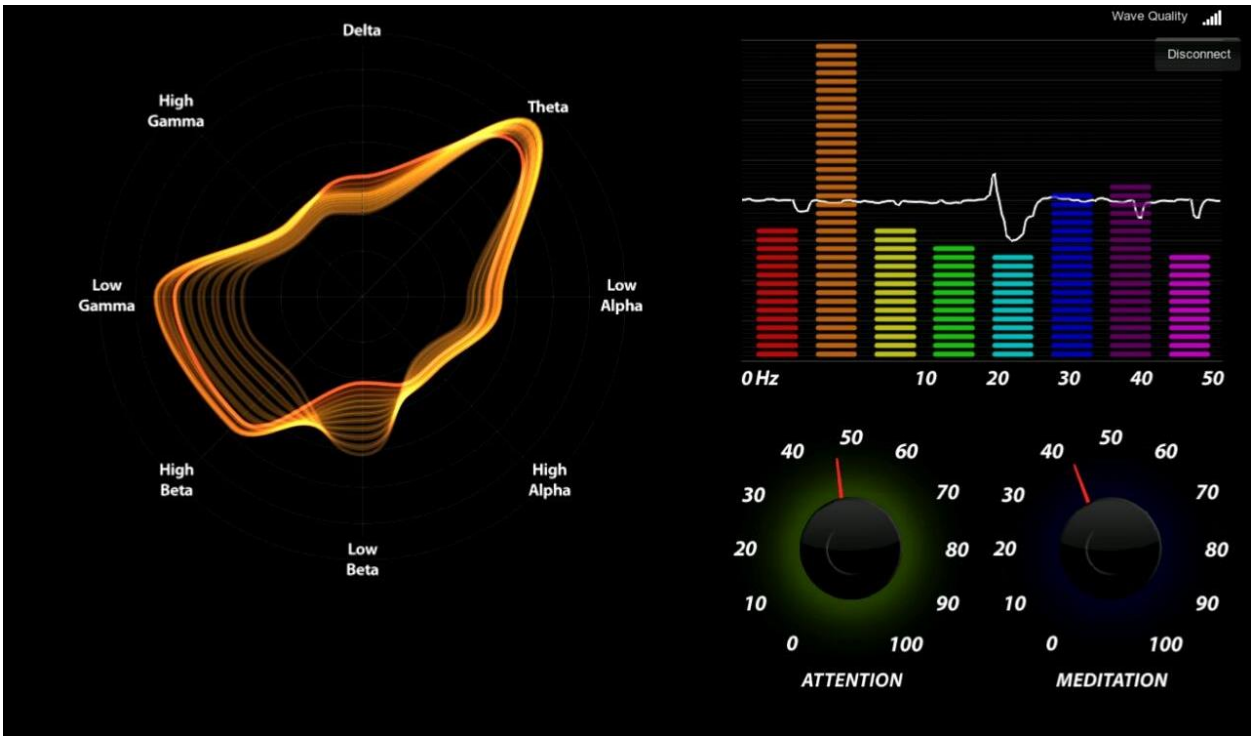

Figura 3-3 Visualizador Mindwave Mobile.

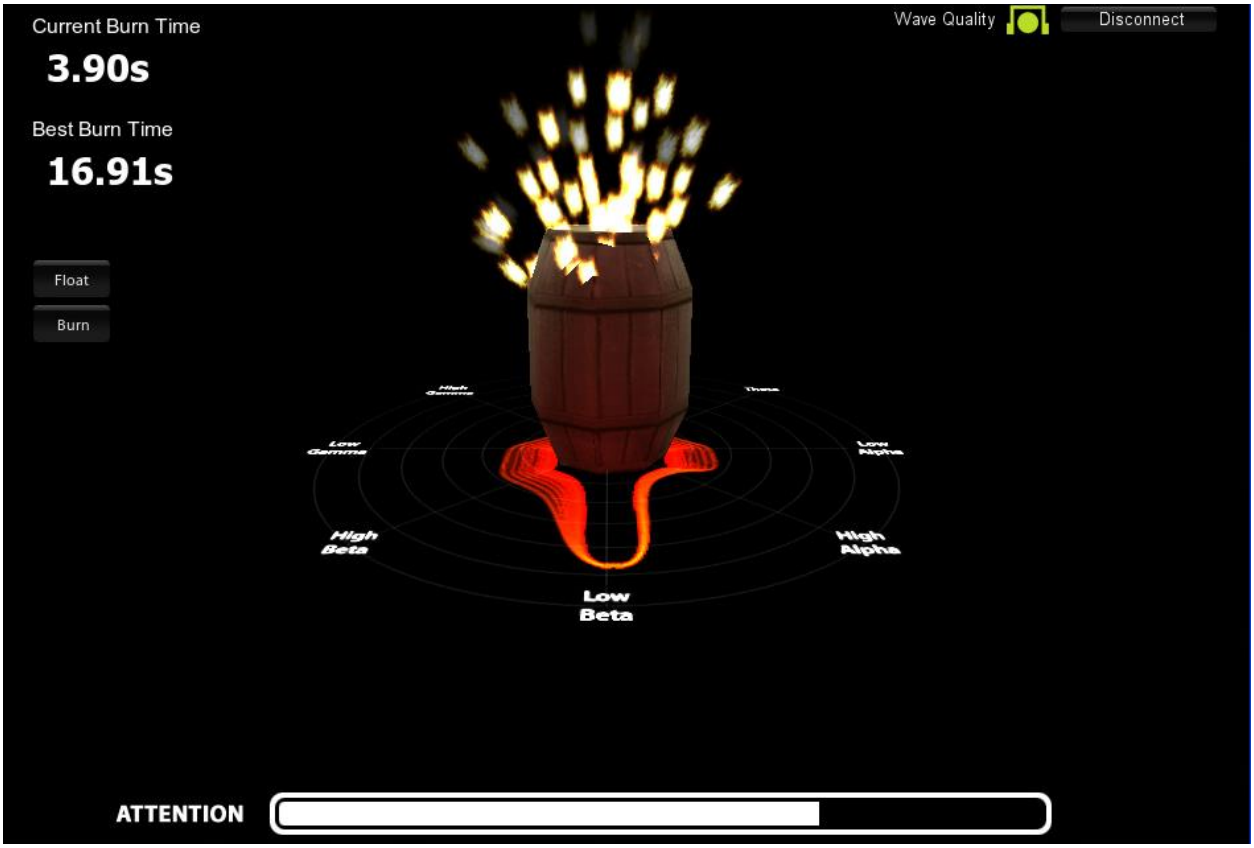

Figura 3-4 Juego mental explosión del barril.

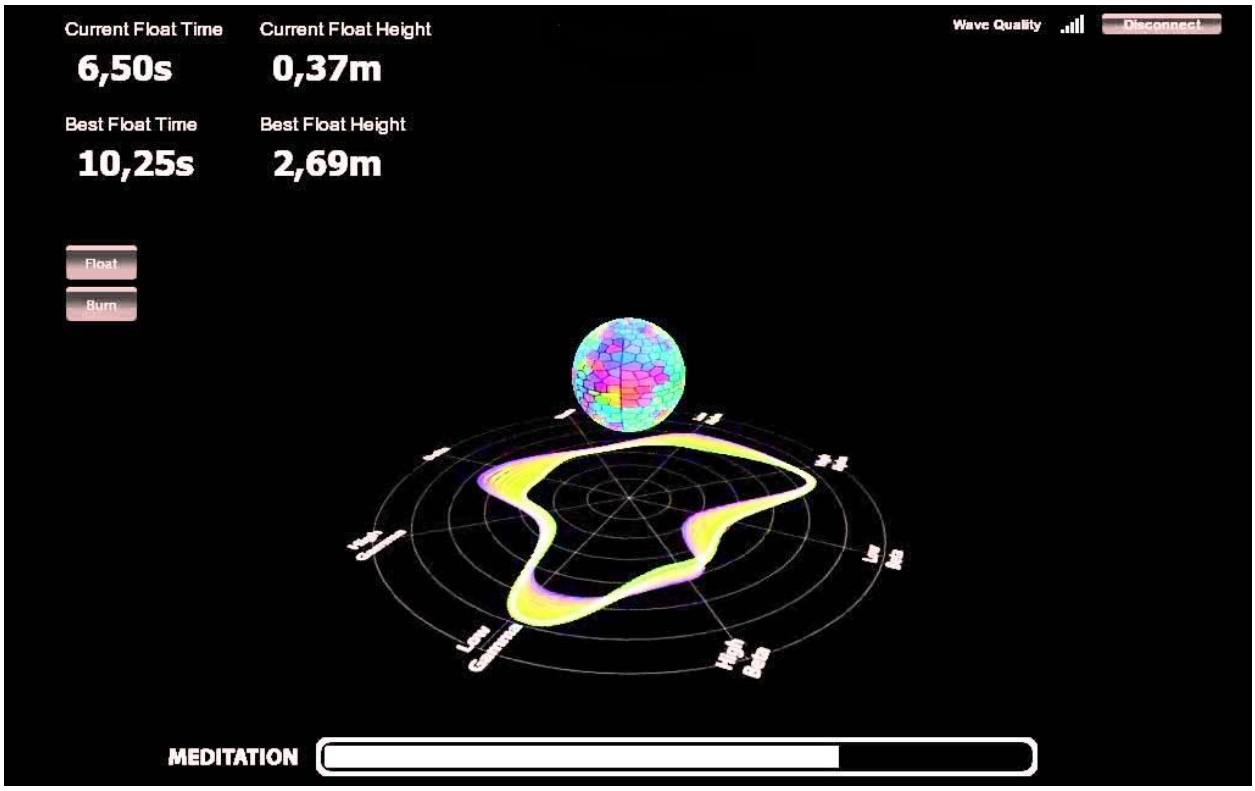

Figura 3-5 Juego mental bola levitando.

El aparato funciona con una pila AAA.

Las medidas que realiza son [16]:

- Ondas cerebrales.
- Espectro de frecuencia de la EEG (alfa, beta…).
- Procesamiento y salida de los niveles de atención y meditación.
- Señal de la EEG que indica el nivel de batería o la calidad de la señal.
- Parpadeo.

Las características físicas son:

- Peso: 90 gramos.
- Medidas: alto 225mm x ancho 155mm x profundidad 165mm.

Las características del bluetooth son:

- Versión: 2.1.
- Potencia de salida: Clase 2.
- Voltaje mínimo: 1 V.
- Rango: 10 metros.
- Consumo de potencia: 80mA (conectado y transmitiendo).
- Indicador de batería baja: 1.1V
- UART (serie): VCC, GND, TX, RX.
- Tasa UART: 57600 baudios.

4 MICROPROCESADOR ARM

4.1 Introducción

La arquitectura ARM es de tipo RISC (Reduced Instruction Set Computer), de 32 bits [17] y actualmente también cuenta con una versión de 64 bits. En sus inicios fue diseñada por *Acorn Computers* [18] para su uso en ordenadores. Ya que trabajan con arquitectura RISC, estos procesadores pueden ser implementados en dispositivos pequeños, como móviles, ya que su consumo es menor y podría utilizar la energía proveniente de una batería. Con lo que los costes son menores también. Debido a su simplicidad se han convertido en elementos muy útiles en aplicaciones de baja potencia. De hecho en la actualidad alrededor del 98% [17] de los teléfonos móviles utilizan al menos un procesador ARM.

La arquitectura es licenciable, por lo tanto, dichas licencias pueden ser usadas para fabricar microcontroladores con este núcleo. En este proyecto se utilizará el microcontrolador STM32F407VG [19]. En concreto la placa STM32F4-DISCOVERY [20]. Dicho microcontrolador utiliza la familia Cortex con núcleo M4 de ARM, que se estudiará a continuación.

4.2 Cortex-M4

4.2.1 Introducción

El núcleo Cortex-M4 es un procesador embebido de alto rendimiento desarrollado para hacer frente a los mercados de control de señales digitales, de fácil uso de las capacidades de procesamiento de señal y control.

La combinación de la funcionalidad de procesamiento de señales de alta eficiencia con el bajo consumo de energía, de bajo coste y los beneficios de facilidad de uso de la familia Cortex-M de procesadores está diseñado para satisfacer la categoría emergente de soluciones flexibles dirigidos específicamente al control del motor, de la automoción, la administración de energía, los mercados de automatización industrial y de audio embebido.

El Cortex-M4 es conceptualmente igual al Cortex-M3 pero añadiendo las instrucciones de DSP y opcionalmente la unidad de punto flotante (FPU), en cuyo caso se denominaría Cortex-M4F.

4.2.2 Arquitectura

El procesador Cortex-M4 está construido sobre un núcleo de procesador de alto rendimiento, con 3-stage pipeline de arquitectura Harvard, que es perfecto para las exigentes aplicaciones embebidas. El procesador ofrece una excepcional eficiencia energética debido al conjunto de instrucciones rápidas y del diseño optimizado, proporcionando hardware de procesamiento de gama alta incluyendo opcionalmente FPU.

Para facilitar el diseño de dispositivos sensibles a los costes, el procesador Cortex-M4 implementa los componentes de forma que se reduce el área de procesador mientras que mejora significativamente las interrupciones y depuración del sistema. El conjunto de instrucciones Cortex-M4 ofrece el notable rendimiento de una arquitectura de 32 bits, con la alta densidad de código de microcontroladores de 8 bits y 16 bits [22].

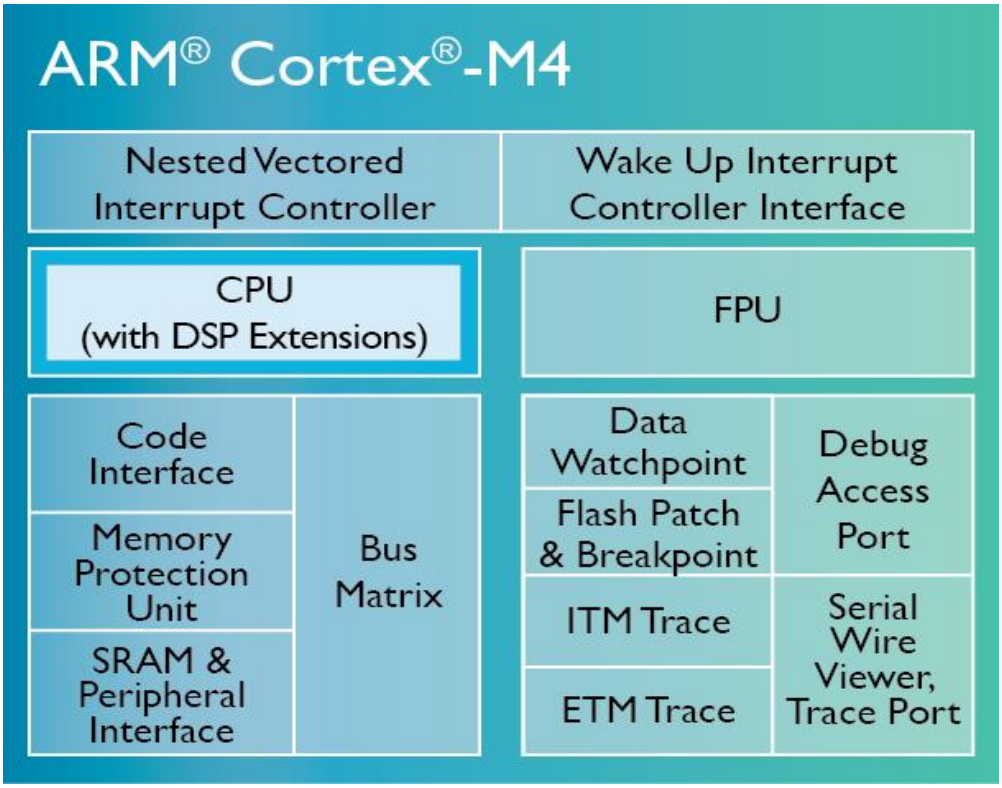

Figura 4-1 Estructura del Procesador Cortex-M4 [21].

4.2.3 Modelo de memoria

El procesador tiene un mapa de memoria fijo, que dispone de 4 GB de memoria direccionable. El tipo de memoria y los atributos determinan como se accede a esa parte.

El orden de llegada de las instrucciones no tiene por qué ser el orden en que se accede a la memoria. Esto es debido a que el procesador puede reordenar las instrucciones para mejorar la eficiencia, a que el procesador tiene múltiples buses, que la memoria o los dispositivos pueden tener diferentes estados de espera o que algunos accesos de memoria se gestionan mediante buffers.

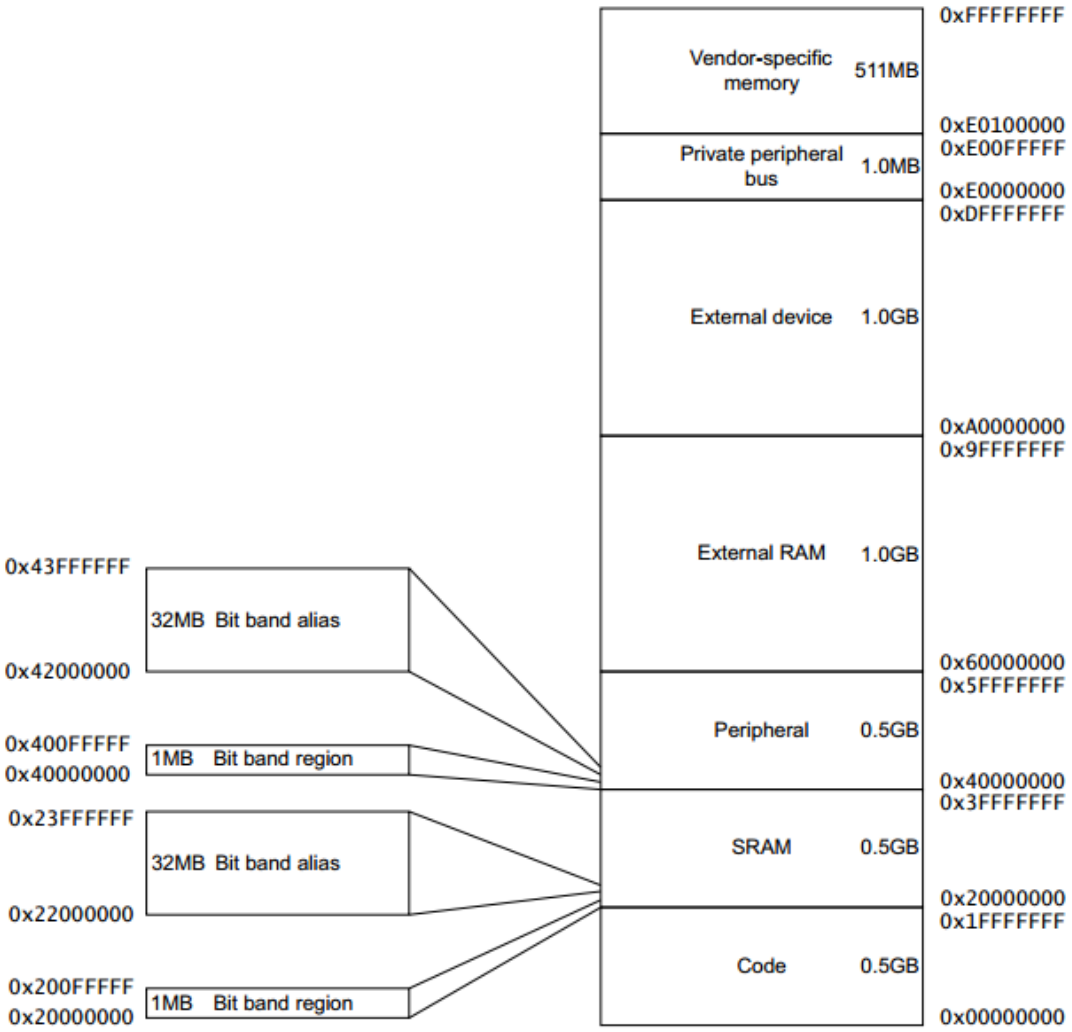

Figura 4-2 Mapa de memoria Cortex-M4 [22].

4.2.4 Características

El procesador Cortex-M4 incorpora [23]:

- Un controlador de vector de interrupciones anidadas (NVIC), integrado en el núcleo del procesador. Las características que incluye son:
  - Interrupciones externas, configurables de 1 a 240.
  - Bits de prioridad, configurables de 3 a 8.
  - Priorización dinámica de interrupciones.
  - Apoyo a la cola de interrupciones y la llegada tardía de las mismas. Esto permite el procesamiento de dichas interrupciones mientras se van actualizando.
  - Un controlador de despertador de interrupciones opcional (WIC).

- Múltiples interfaces de buses de alto rendimiento.
    - Tres interfaces de alto rendimiento (AHB).
    - Bus de periférico privado (PPB) basado en una interfaz de bus de periférico avanzado (APB).
    - Acceso a la memoria de alineación.
    - Buffer de escritura para el buffer de escritura de datos.
    - Acceso exclusivo de las transferencias para los sistemas de multiprocesador.

- Un depurador de bajo coste con capacidad de crear puntos de interrupción, implementar wacthpoints, rutas y depuración de ayuda mediante printf.
    - Acceso de depuración a toda la memoria y registros del sistema, incluyendo acceso a dispositivos del mapa de memoria y acceso a los registros internos del núcleo cuando estén parados.
    - Puerto de depuración Serial Wire (SW-DP) o Puerto de depuración JTAG (JTAG-DP).

- Una protección de memoria opcional (MPU).
    - Ocho regiones de memoria.
    - Desactivación de subregiones, lo que permite el uso eficiente de regiones de memoria.

- Una unidad de punto flotante (FPU).
    - Instrucciones de 32 bits de precisión simple.
    - Instrucciones de acumulación y multiplicación combinadas para incrementar la precisión.
    - Soporte hardware para la conversión, suma, resta, multiplicación con acumulador opcional, división y raíz cuadrada.
    - 32 registros dedicados de 32 bits de precisión simple, direccionables también como 16 registros de palabra doble (double-word).

4.3 STM32F407VG

4.3.1 Introducción

El microprocesador STM32F407VG se basa, como se ha comentado anteriormente, en el procesador de alto rendimiento ARM Cortex-M4 de 32 bits con arquitectura RISC, operando a una velocidad de 168 MHz. Incluye además memorias embebidas de alta velocidad (memorias Flash) de 1 Mbyte y hasta 192 Kbytes de SRAM y múltiples puertos de entrada/salida conectados a dos buses APB, tres AHB y una matriz de buses multi-AHB de 32 bits.

Todos los dispositivos incorporan tres convertidores analógico/digital de 12 bits, dos convertidores digital/analógico, un RTC de baja potencia, 12 TIMERs de propósito general de 16 bits, incluidos dos TIMERs PWM para el control de motores y dos TIMERs de propósito general de 32 bits.

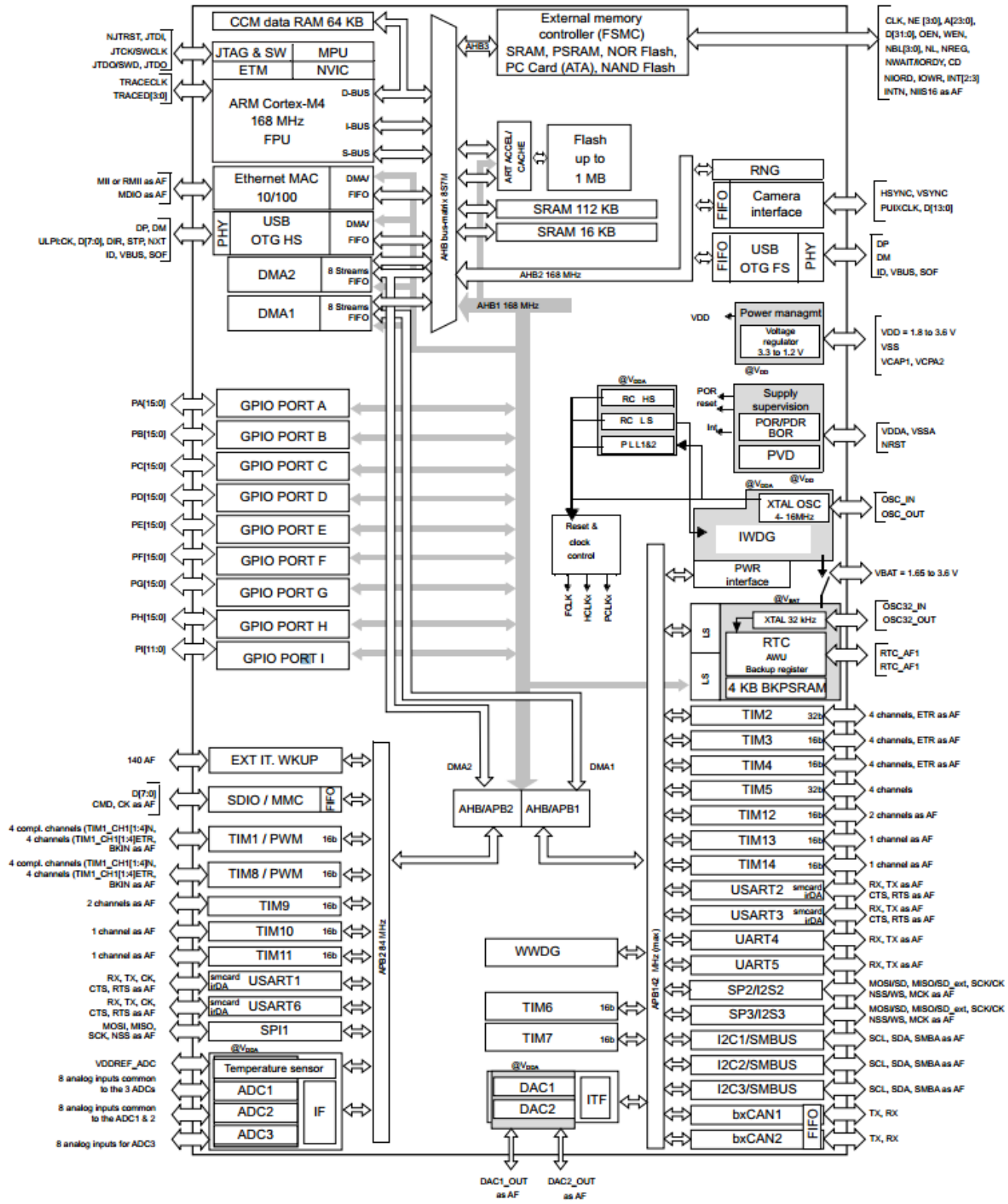

Figura 4-3 Diagrama de bloques STM32F407Vx [24].

4.3.2 Características

- Núcleo Cortex-M4 de ARM, frecuencia de hasta 168 MHz, MPU e instrucciones para DSP.
- Memorias:
    - Hasta 1 Mbyte de memoria Flash.
    - Hasta 192 + 4 Kbytes de SRAM.
    - Controlador de memoria estática para el soporte de Flash compactas, SRAM, PSRAM, NOR y NAND.
- Interfaz LCD.
- Reloj, reset y control de alimentación:
    - 1.8 V a 3.6 V para alimentación de aplicaciones y puertos entrada/salida.
    - Oscilador de 4 a 26 MHz de cristal.
    - Oscilador de 32 kHz para RTC.
- Modo de operación de baja potencia:
    - Modos de sueño, parada y standby.
- 3 convertidores A/D de 12 bits.
- 2 convertidores D/A de 12 bits.
- DMA de propósito general.
- Hasta 17 Timers.
- Modos de depuración.
- Hasta 140 puertos de entrada/salida con capacidad de interrupción.
- Hasta 15 interfaces de comunicación:
    - Hasta 3 interfaces I2C.
    - Hasta 4 USARTs/2UARTs.
    - Hasta 3 SPIs.
    - 2 interfaces CAN.
    - Interfaz SDIO.
- Unidad de cálculo de CRC.
- RTC.

4.4 STM32F4-DISCOVERY

4.4.1 Introducción

Tras un breve resumen del procesador usado, se va a pasar a explicar el kit de desarrollo que incorpora dicho procesador y que ha sido usado en la realización de este proyecto. El kit en cuestión es la placa de evaluación STM32F4-Discovery [25].

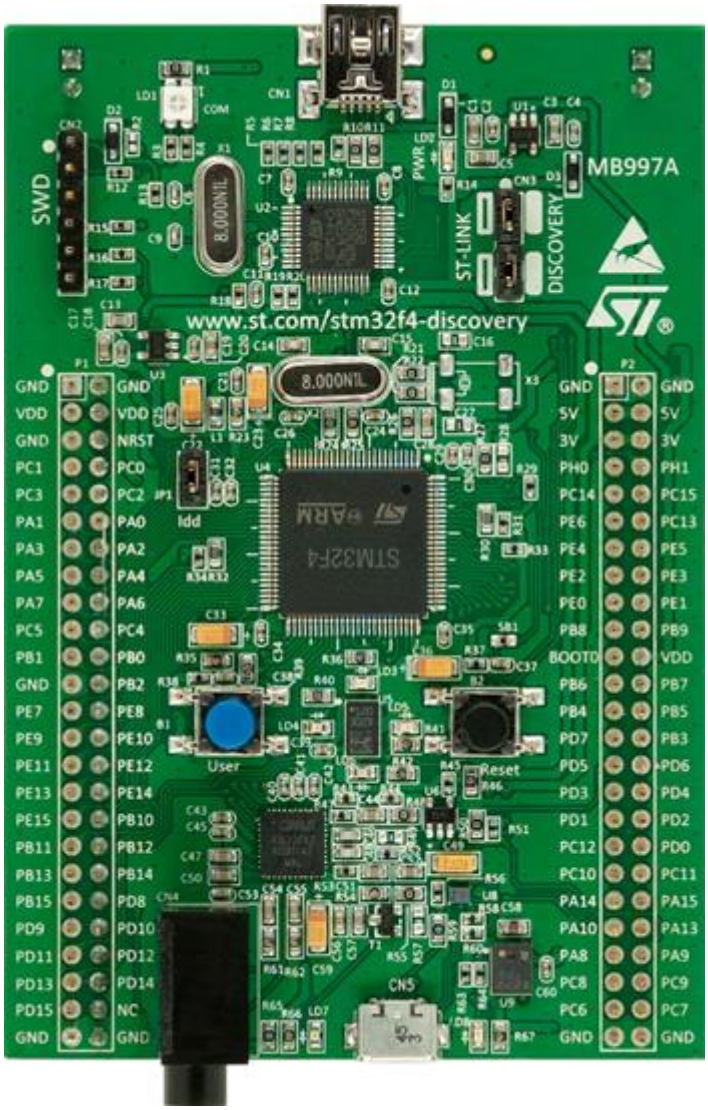

Figura 4-4 Placa de evaluación STM32F4-Discovery [25].

Esta placa permite al usuario el desarrollo de aplicaciones con el procesador STM32F407. Es una herramienta muy útil tanto para principiantes como para expertos en la familia ARM.

La placa incluye la herramienta de depuración ST-LINK/V2, dos acelerómetros, un micrófono digital, un conversor digital/analógico de audio, LEDs, dos botones y un USB-OTG con conector micro-AB.

4.4.2 Características

- Microcontrolador STM32F407VGT6.
- ST-LINK/V2 para depuración.
- Tensión de alimentación de la placa: 5V.
- Tensiones de alimentación para aplicaciones externas: 3V y 5V.
- Acelerómetro de 3 ejes.
- DAC de audio.
- 8 LEDs:
    - LD1 (rojo/verde) para comunicación USB.
    - LD2 (rojo) para alimentación.
    - 4 LEDs de usuario, LD3 (naranja), LD4 (verde), LD5 (rojo), LD6 (azul).
    - 2 LEDs USB-OTG, LD7 (verde) y LD8 (rojo).
- 2 pulsadores: uno de usuario y otro de reset.
- Software libre integrado, que incluye una variedad ejemplos.

4.4.3 Requisitos del Sistema

Para desarrollar alguna aplicación en la placa de evaluación Discovery hay que conectarla como se muestra en la Figura 4-5.

Los requisitos mínimos para desarrollar cualquier aplicación son:

- PC con Windows (2000, XP, Vista, 7).
- USB Tipo A a Mini-B, usado para alimentar la placa y conectar el depurador ST-LINK/V2.

Algunas aplicaciones requieren de hardware adicional:

- USB Tipo A a Micro-B, para conectar la placa a otro dispositivo.

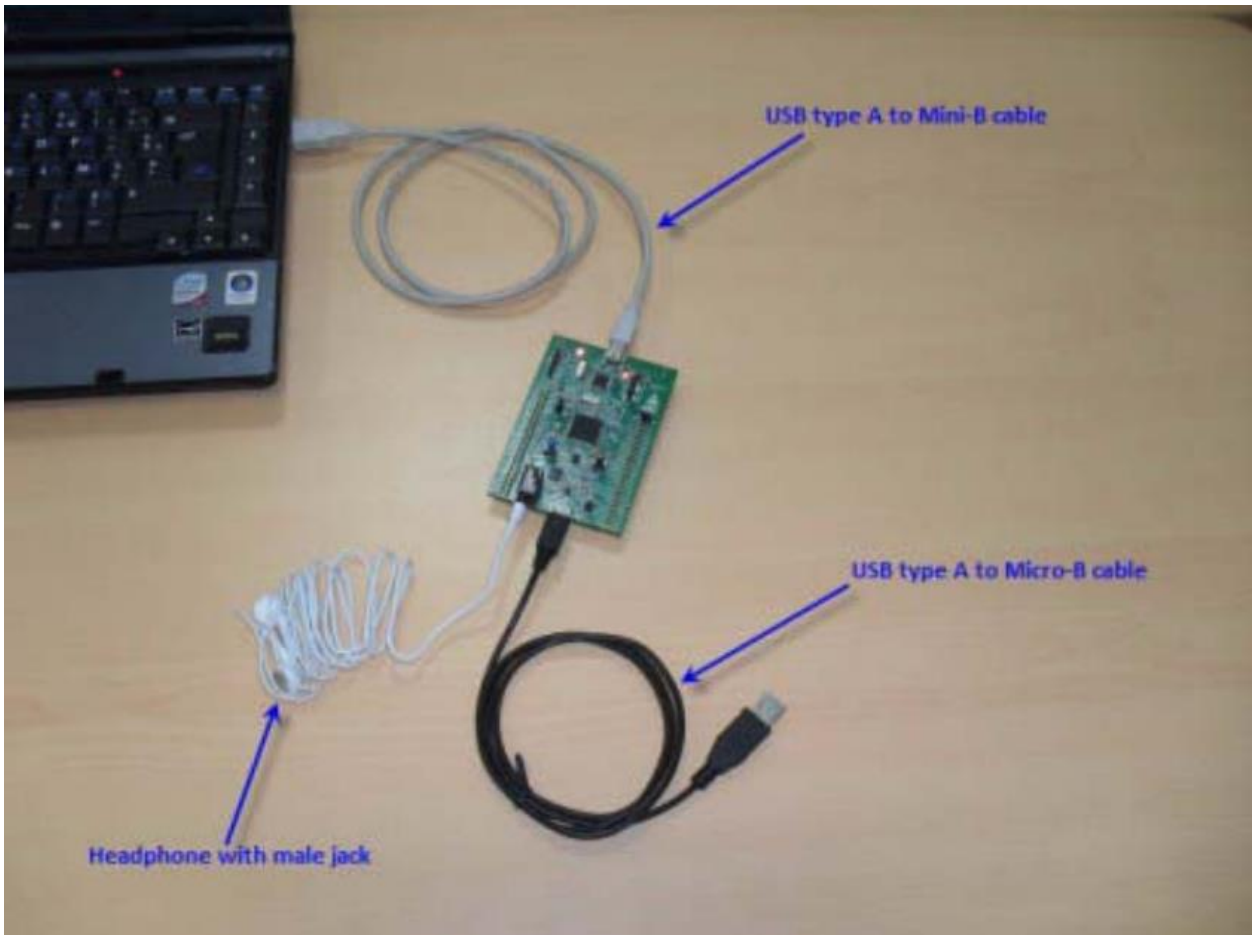

Figura 4-5 Entorno hardware Discovery.

4.5 Entorno de desarrollo

4.5.1 Introducción

Se pueden encontrar diferentes entornos de desarrollo para la placa de evaluación STM32F4-Discovery: IAR Embebed Workbench for ARM, MDK-ARM Keil UVision, Atollic TrueSTUDIO o Tasking.

En este caso concreto, se ha usado el entorno de desarrollo Keil UVision, cuyo entorno de trabajo se muestra en la siguiente Figura 4-6.

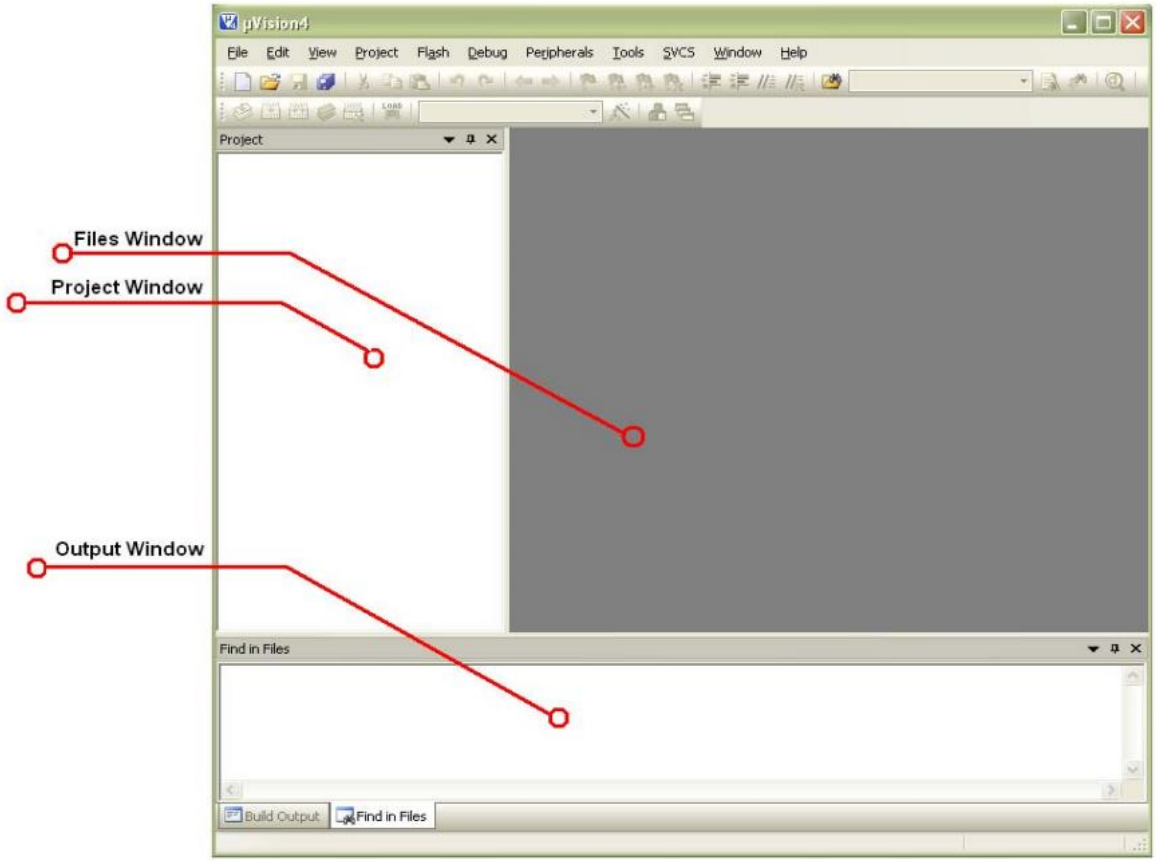

Figura 4-6 Entorno Keil.

4.5.2 Proyecto Nuevo

En primer lugar se creó un proyecto nuevo para comenzar a trabajar sobre la placa. Esta operación se muestra en la Figura 4-7.

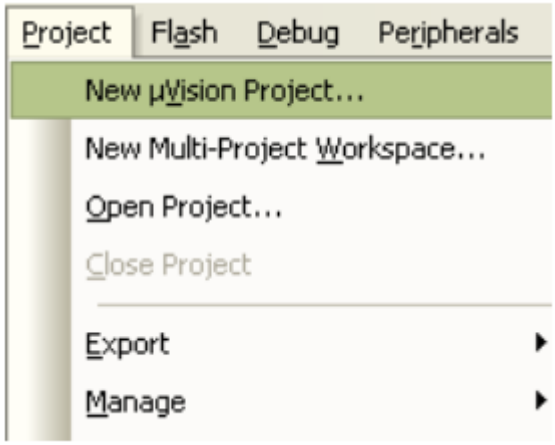

Figura 4-7 Nuevo proyecto Keil.

Una vez que se ha guardado el proyecto con el nombre que se desea, hay que seleccionar la placa de desarrollo a utilizar.

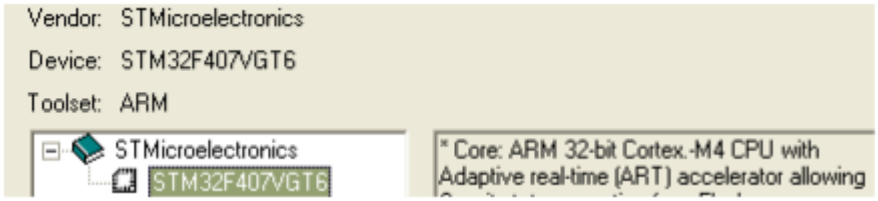

Figura 4-8 Selección de placa Keil.

Una vez seleccionada la tarjeta, saldrá un menú por el cual podemos seleccionar las interfaces o componentes de la placa que se van a utilizar, de forma que no se utilice espacio de memoria que no se vaya a aprovechar.

Tras esto, hay que configurar las opciones de la tarjeta de desarrollo usada.

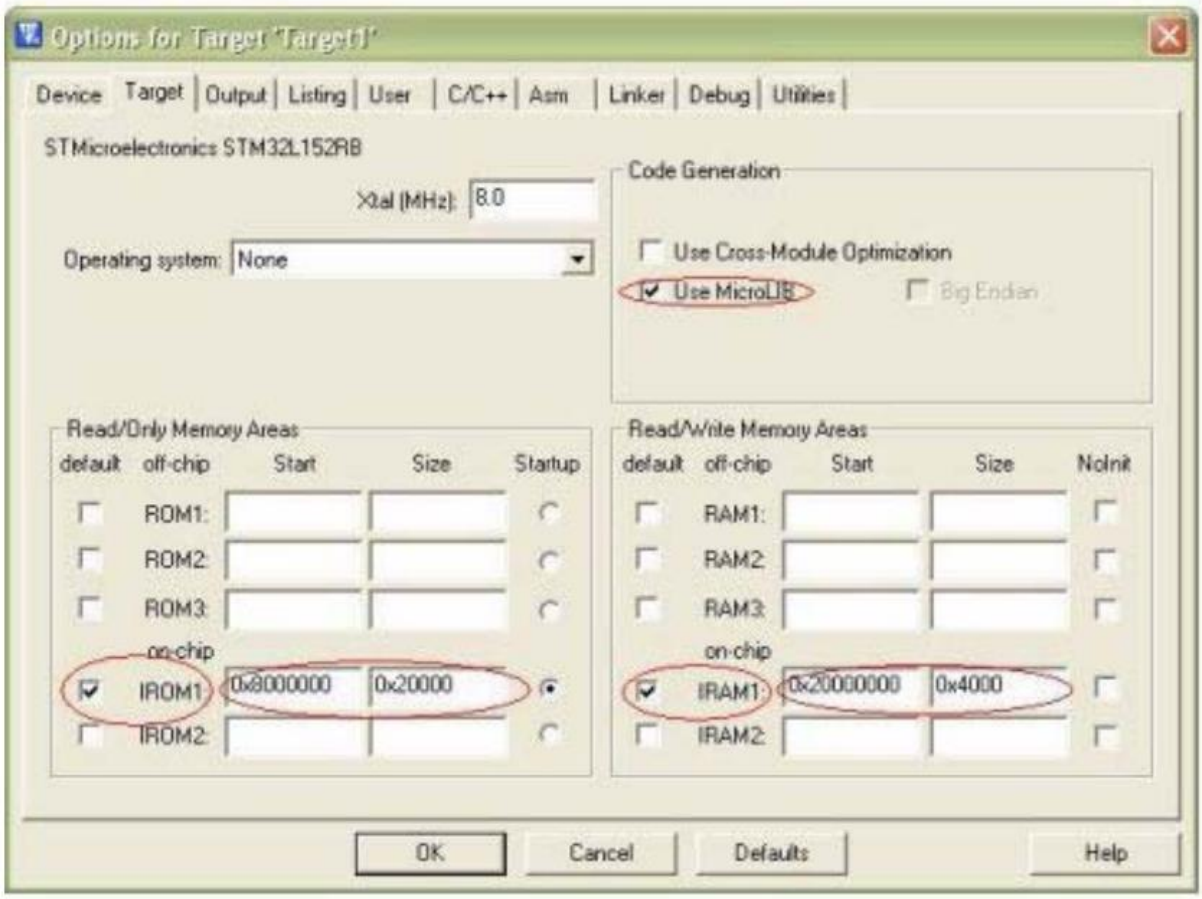

Figura 4-9 Opciones tarjeta Keil 1.

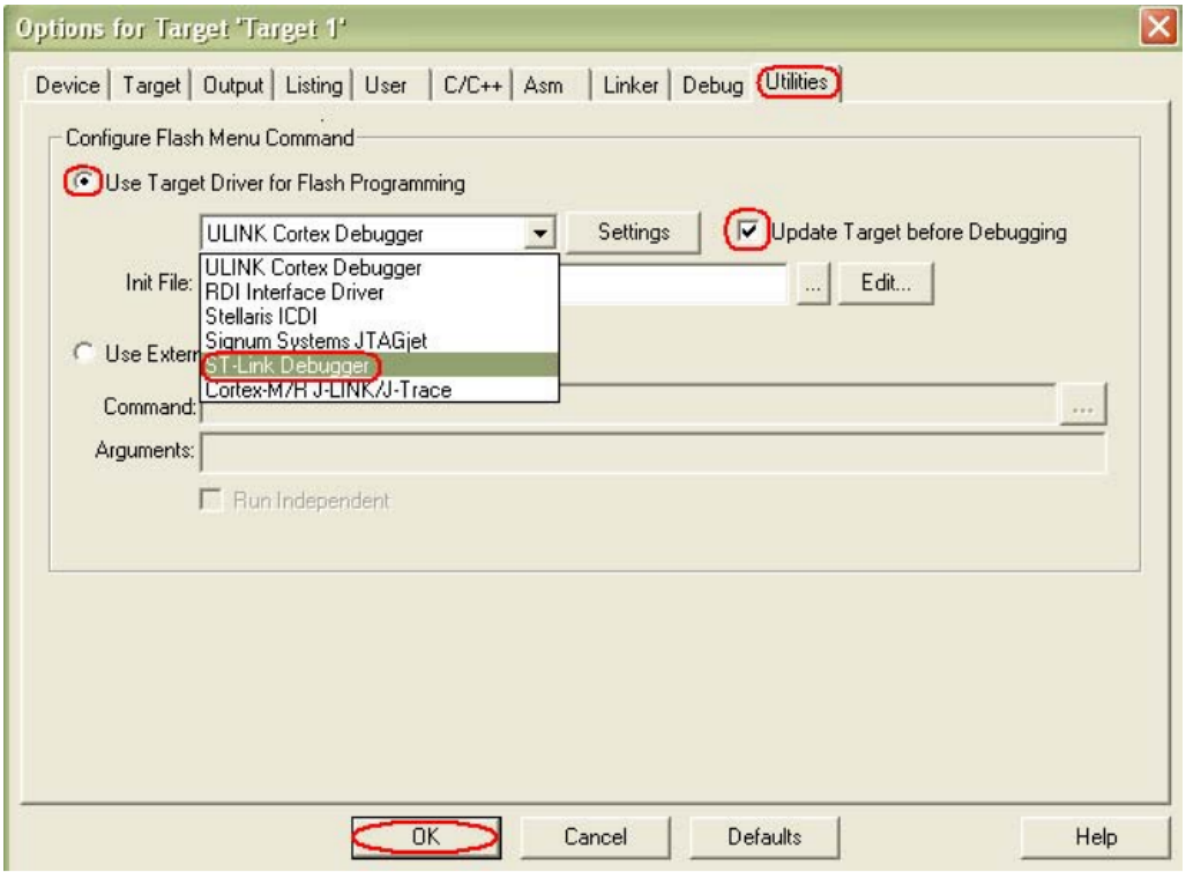

Figura 4-10 Opciones tarjeta Keil 2.

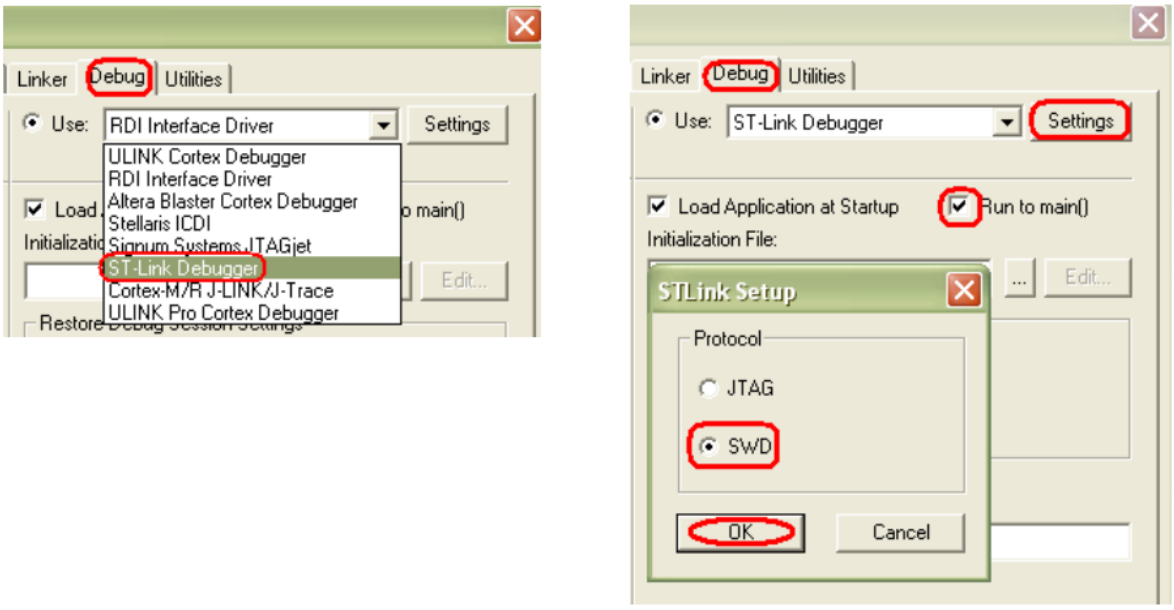

Figura 4-11 Opciones tarjeta Keil 3.

Una vez configurada la tarjeta se añaden los ficheros necesarios que forman el proyecto.

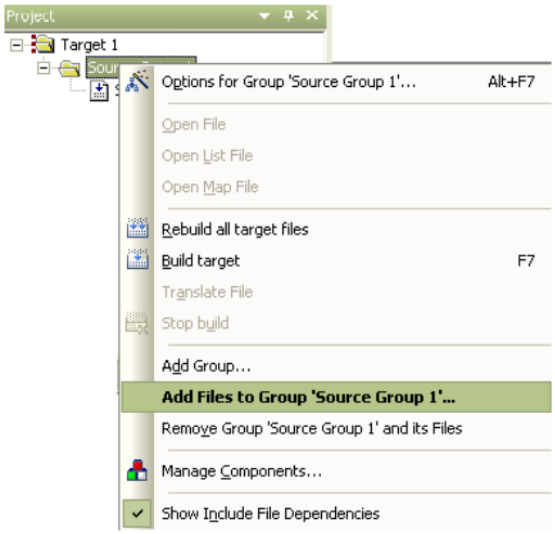

Figura 4-12 Añadir ficheros a proyecto Keil.

Se pueden, y de hecho se deben, crear ramas dentro del directorio del proyecto para dividir los ficheros según la categoría. Es decir, se pueden guardar en un apartado los ficheros que sean drivers, en otro las aplicaciones, en otros los módulos utilizados, etc…

La estructura de este proyecto se muestra a continuación.

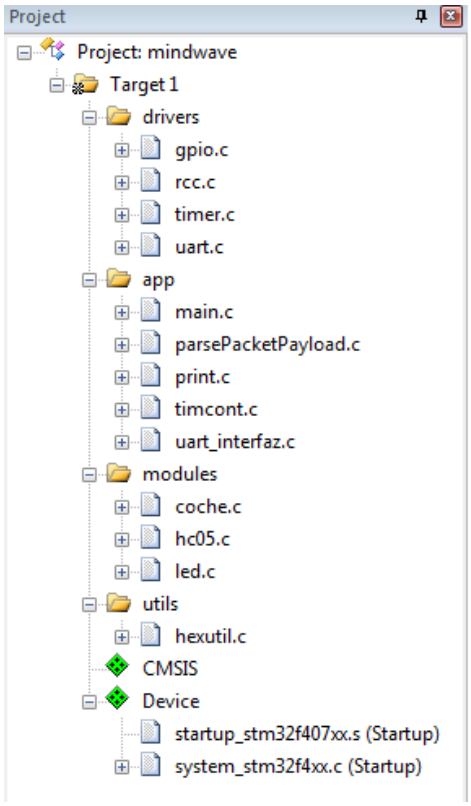

Figura 4-13 Directorio del proyecto.

Cuando se tenga el proyecto terminado, habrá que realizar la compilación del proyecto. Primero para ver posibles errores y/o warnings y segundo para poder pasar el ejecutable a la placa y probarla.

Se puede depurar el programa para refinarlo hasta el punto que se quiera o para corregir fallos. A continuación se explica el modo de depuración del entorno Keil.

4.5.3 Depuración del Proyecto

Para depurar el proyecto hay que pasar a la ventana o modo de depuración.

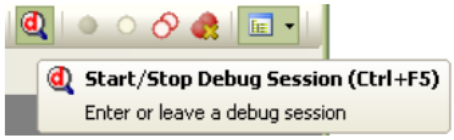

Figura 4-14 Iniciar sesión de depuración.

Se abrirá una ventana como la siguiente.

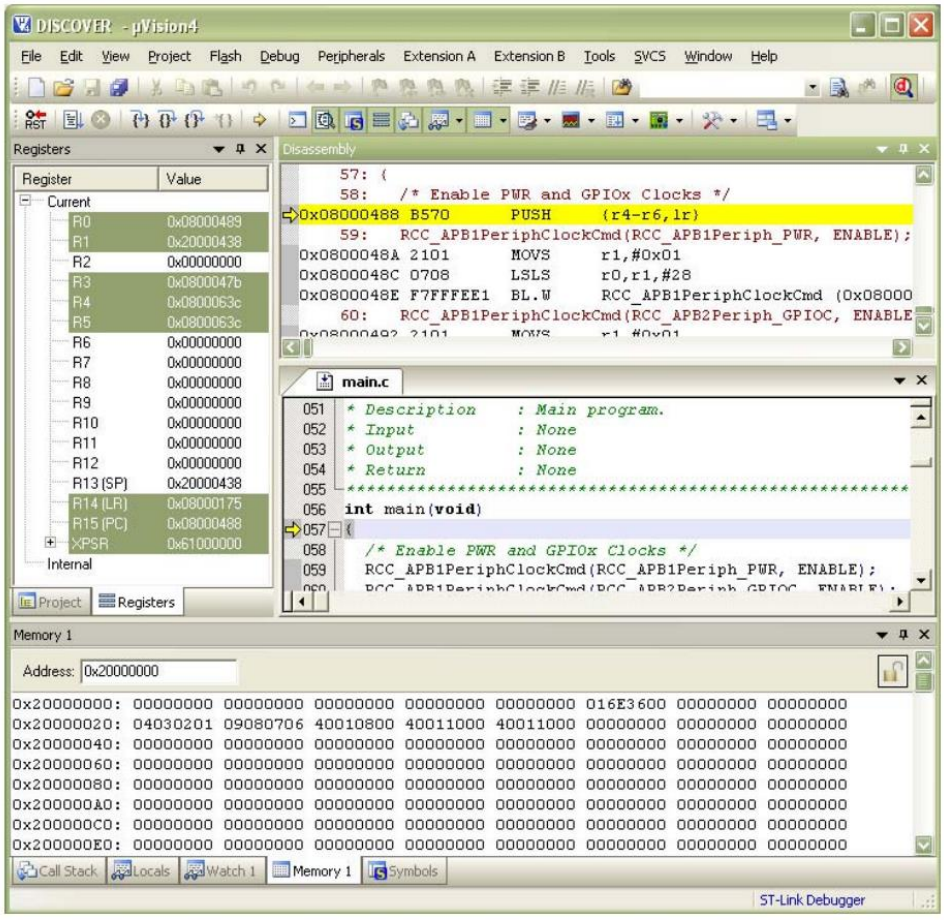

Figura 4-15 Ventana de depuración.

En el modo de depuración se pueden habilitar y establecer puntos de parada, para ir parando el código e ir viendo posibles fallos a medida que va trabajando el programa. Se pueden visualizar variables, registros y eventos durante la ejecución del código.

5 APLICACIÓN

5.1 Introducción

Una vez explicados todos los componentes necesarios para desarrollar este proyecto, a falta de hacer el estudio del coche teledirigido empleado, se va a detallar el proceso y la funcionalidad del proyecto paso a paso. Empezando por el casco Mindwave y como se conecta al microprocesador, para seguidamente parsear los paquetes recibidos por el casco, que también se explicarán, así como el procesamiento de las tramas recibidas para crear la funcionalidad específica de este trabajo. Finalmente se detallará el coche usado a nivel electrónico y como se transmiten las órdenes de la placa al coche.

5.2 Conexión MindWave con STM32F4-Discovery

En primer lugar, hay que decir que para conectar a través de bluetooth el casco con la placa de desarrollo, se ha necesitado un módulo bluetooth *hc-05 [26]* que usa la Discovery.

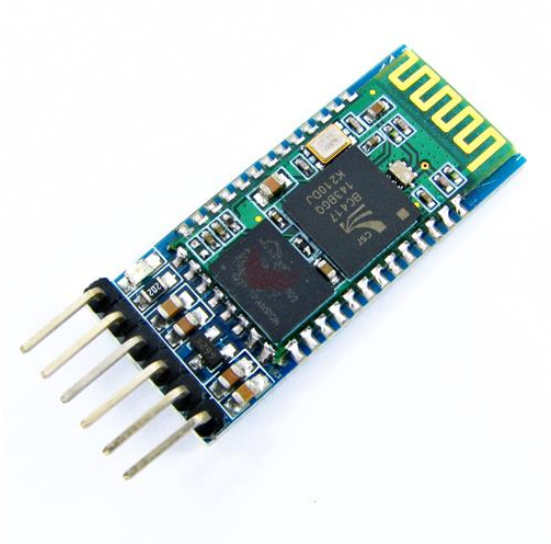

Figura 5-1 Modulo bluetooth HC-05.

El modulo bluetooth se configura con una serie de comandos, comandos AT. Está cualificado con la versión 2.0 de bluetooth y tiene las siguientes especificaciones:

- Baja potencia de operación: 1.8V. 1.8V a 3.6V para entrada/salida.
- Interfaz UART con tasa de baudios programable.
- Antena integrada.

- Por defecto:
    - 38400 baudios.
    - 8 bits de datos.
    - 1 bit de parada.
    - Sin paridad.
- Tasas soportadas: 9600,19200, 38400, 57600, 115200, 230400, 460800.
- Dando un pulso alto en el puerto PIO0, el dispositivo se desconectará.
- Por defecto se conecta al último dispositivo pareado.
- Pin por defecto para el pareado: “0000”.

El modulo posee 34 pines. En este caso vamos a utilizar 6:

- VCC a 3.3 V.
- GND.
- Pin 1: UART_TX.
- Pin 2: UART_RX.
- Pin 11: Reset.
- Pin 34: Key/Enable.

El proceso para la conexión del casco se describe en los siguientes pasos:

1. Se configura el módulo de forma que trabaje con comandos AT. Se pone a 1 el pin 34 (Enable) y se hace un reset del módulo. Se configura la USART del micro a 38400 baudios y se empiezan a enviar los comandos.
2. Como por defecto el casco tiene el pin “0000”, se configura el mismo pin al módulo (aunque por defecto es la clave que usa, por seguridad, se repite la orden): “AT+PSWD=0000\r\n”. Se espera el “OK\r\n” que se debe recibir después de cada comando AT para asegurarse de que todo funciona correctamente. Esto último se realizada cada vez que se manda un comando.
3. A continuación se configura el rol del módulo. En este caso se configura como maestro (master): “AT+ROLE=1\r\n”.
4. Seguidamente, como vamos a conectarnos a una dirección física fija, la del casco, se manda el siguiente comando: “AT+CMODE=0\r\n”.
5. Ahora se configura el modulo a la tasa que funciona el casco por defecto, 57600 baudios: "AT+UART=57600,1,0\r\n".
6. Por último se manda el comando con la dirección del casco, para establecer la conexión: "AT+BIND=2068,9d,4c14f0\r\n".

Una vez establecida la conexión, el led del casco se quedará de color azul fijo.

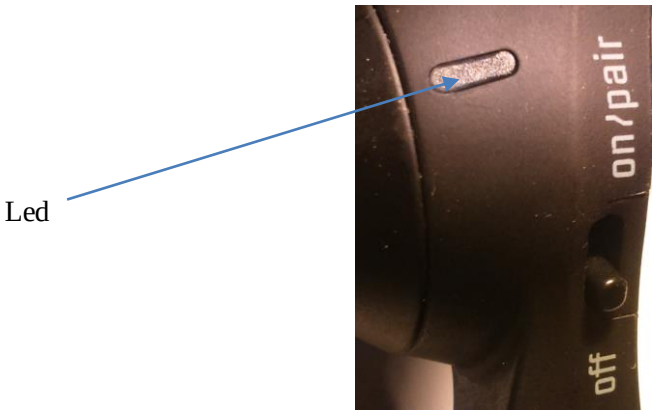

Figura 5-2 Led MindWave.

Cabe destacar que para realizar la conexión, en primer lugar hay que poner el casco en modo emparejamiento. Esto se realiza dejando el botón en la posición final durante 2 segundo seguidos y posteriormente soltarlo al medio. En la siguiente figura se puede observar el funcionamiento descrito.

Figura 5-3 Funciones del botón de MindWave.

Tras la conexión establecida se entra en modo comunicación, pero antes se ha puesto un Timer de unos 14 segundos, hasta que el programa empiece a funcionar. Este retraso se mete para que se estabilice la señal que proviene del casco. Durante ese tiempo, en la Discovery, se podrá observar un led rojo parpadeando hasta que termina la cuenta. Una vez finalizado los 14 segundos, el led se apagará y es entonces cuando se empiezan a parsear los paquetes.

5.3 Parseo de los paquetes

5.3.1 Datos enviados por MindWave

EL casco o diadema de ondas cerebrales puede enviar diferentes tipos de información en sus paquetes. A continuación se van a detallar cada una de las medidas utilizadas en este trabajo que MindWave transmite.

- POOR_SIGNAL Quality: calidad de la señal. Esto se manda en un byte sin signo y describe como de pobre es la medida de la señal. El rango va desde 0 a 255. A mayor número, mayor es el ruido detectado. En concreto, el valor 200 indica que no hay contacto entre el sensor y la frente del usuario. Esta medida se saca una vez por segundo.

- ATTENTION eSense: un byte sin signo, que contiene el nivel de atención o concentración del individuo que porta la diadema. El rango de esta medida es de 0 a 100. Valores de 1 a 40 indican un nivel muy bajo de concentración, entre 40 y 60 se considera neutral, de 60 a 80 se empieza a considerar un poco elevado, mientras que finalmente valores entre 80 y 100 son considerados elevados. Esta señal es transmitida una vez por segundo.

- MEDITATION eSense: también es un byte sin signo, que indica el nivel de calma o relajación. Los valores van de 0 a 100, tal como sigue el nivel de ATTENTION. A su vez, esta medida se envía cada segundo.

- RAW Wave Value: este valor consiste en dos bytes y representa una muestra de onda cerebral. Su valor se encuentra en una variable de 16 bits con signo, cuyo rango va de -32768 a 32767. El primer byte representa los bits más altos del complemento a2, mientras que el segundo byte representa los bits más bajos. Para reconstruir la señal hay que desplazar el primer byte 8 bits y realizar una operación OR con el segundo: ***Raw = (byte_0 << 8) | byte_1;*** dónde, byte_0 es el más bajo y byte_1 el más alto. Esta señal es transmitida por el casco 512 veces por segundo.

Nombre	Nº de bytes	Rango de Valores	Tasa de envío
POOR_SIGNAL	Unsigned byte	0 – 255	1 x segundo
ATTENTION	Unsigned byte	0 – 100	1 x segundo
MEDITATION	Unsigned byte	0 – 100	1 x segundo
RAW Wave Value	2 Signed bytes	-32768 - 32767	512 x segundo

Tabla 5-1 Cuadro resumen de datos enviados por MindWave.

5.3.2 Estructura de los paquetes

Una vez establecida la conexión, el micro empieza a recibir los paquetes que el casco manda. Dichos paquetes son enviados de forma asíncrona en serie y en cadena de bytes. Se deben parsear correctamente los paquetes para interpretar los valores que contiene, explicados anteriormente.

Cada paquete está formado por 3 partes:

- Cabecera.
- Carga útil.
- Checksum.

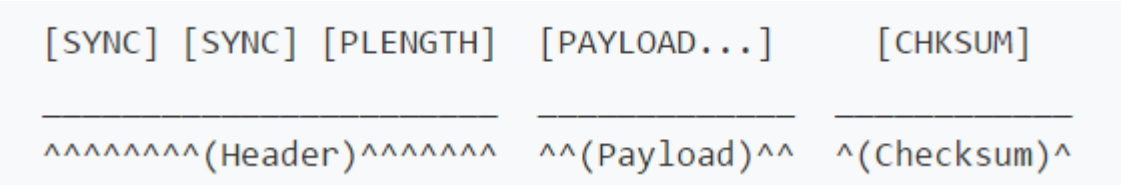

Figura 5-4 Estructura de los paquetes de MindWave [27].

En total habrá 4 bytes más la carga útil. Por lo tanto, como mucho, la longitud de los paquetes será de 173 bytes, ya que "SYNC", "PLENGTH" y "CHECKSUM" tienen una longitud de 1 byte cada uno y "PAYLOAD" como máximo puede tener una longitud de 169 bytes.

1. Cabecera: está formada por 3 bytes. Dos bytes "SYNC" y uno "PLENGTH". Los dos primeros bytes de sincronización deben ser "0xAA". Se envían dos seguidos para evitar fallos en la recepción. El byte "PLENGTH" indica la longitud, en bytes, de la carga útil (PAYLOAD). Puede valer entre 0 y 169. Cualquier valor mayor, indicaría que se ha producido un error.
2. Carga útil: Payload, es un conjunto de bytes. La interpretación de los datos enviados en la carga útil se explicarán más adelante, en el parseo de las tramas. Para dicho parseo, primero tiene que haberse dado por válido el paquete tras la comprobación del Checksum.
3. Checksum: este byte se usa para comprobar la integridad de los paquetes. El Checksum se define como la suma de todos los bytes del paquete dentro de la carga útil, seguidamente se toman los 8 bits más bajos de la suma y se realiza el complemente a1. Es decir, se debe realizar esta operación con cada paquete y comprobar que el valor resultante coincide con el valor del byte "CHECKSUM". Si no coinciden los valores, el paquete se descarta. Por el contrario, si coinciden, el paquete se da por bueno y se pasa al parseo de la carga útil.

5.3.3 Parseo de los paquetes paso a paso

Seguidamente se va a realizar una guía de la decodificación de los paquetes paso a paso, tal como se sigue en el código implementado:

1. Se van leyendo los bytes recibidos hasta encontrar un 0xAA, byte de sincronización.
2. Se lee el siguiente byte y se comprueba que es también 0xAA:
    - Si no lo es, se vuelve al paso 1.
    - En otro caso se pasa al punto 3.
3. Se extrae el siguiente byte, que será PLENGTH:
    - Si el byte contiene el valor 170, se repite el punto 3.
    - Si contiene un valor mayor de 170, se vuelve al paso 1.
    - En cualquier otro caso, se continúa por el punto 4.
4. Se leen los bytes (indicados en PLENGTH) de la carga útil, PAYLOAD, y se almacenan en una tabla o vector. A su vez se le suma cada byte que se lee a un acumulador (o variable), que se comprobará con el checksum al final.

5. Se cogen los 8 bits más bajos del acumulador y se invierten:
    - acumulador &= 0xFF;
    - acumulador = ~acumulador & 0xFF;
6. Por último, se lee el siguiente byte de la cadena, el CHECKSUM:
    - Si el valor de CHECKSUM no coincide con el del acumulador, se ha producido un fallo, y se descarta el paquete.
    - En otro caso, se considera como valido dicho paquete y se pasa al parseo de la carga útil.
    - En cualquier caso, se retorna al punto 1.

5.3.4 Parseo de la carga útil o Payload

Como se ha descrito anteriormente, una vez que los paquetes se dan por validos tras comprobar el checksum, se procede a obtener la información que contiene el campo PAYLOAD, carga útil. Para comenzar, se va a explicar la estructura de dicha carga útil.

Payload contiene, una serie de bytes, y se distribuyen como se aprecia en la siguiente figura:

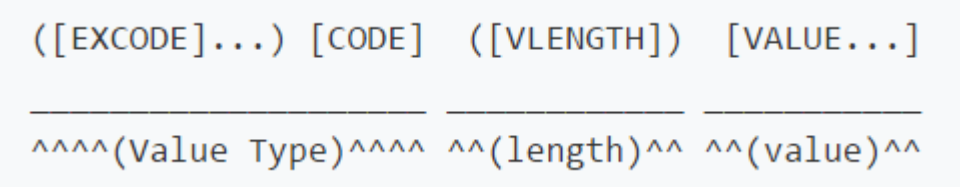

Figura 5-5 Estructura de las tramas [27].

Los bytes entre paréntesis son opcionales. Cada trama debe empezar con cero o más bytes [Extended Code], que contienen el valor 0x55. En este caso, este byte no ha hecho falta, ya que con el byte [Code], se tiene suficiente para obtener la información que se necesita.

El byte [Code], indica el tipo de valor codificado. Por ejemplo, un valor en [Code] de 0x04, indica que el siguiente byte contiene la información relativa con el nivel de atención. Hay que hacer notar que el valor 0xAA no forma parte de la tabla de códigos, que se verá a continuación, para no dar lugar a error, ya que ese valor es el que contienen los bytes de sincronización.

Si el valor de [Code] va de 0x00 a 0x7F, determina que el valor a extraer tiene un byte de longitud. En ese caso no habrá byte [VLength]. El valor a extraer ([Value]) vendrá a continuación del [Code].

Por el contrario, si el valor de [Code] es mayor de 0x7F, en [VLength] se encontrará el número de bytes que tendrá el campo [Value].

A continuación se puede observar una tabla con los códigos y valores que se usan para este proyecto.

[Extended code]	[Code]	[Length] (Bytes)	Valor de los datos
0	0x02	1	POOR_SIGNAL: 0-255
0	0x04	1	ATTENTION: 0-100
0	0x05	1	MEDITATION: 0-100
0	0x80	2	RAW Wave Value: valor con signo big endian de 16 bits en complemento a2. (byte más alto seguido del más bajo): -32768-32767

Tabla 5-2 Tabla de definición de códigos [27].

5.3.5 Parseo de las tramas paso a paso

Como se hiciera con los paquetes, se va a detallar una guía paso a paso, para decodificar la información de las tramas. Esto debe realizarse a todos los bytes de la carga útil, dónde el número de bytes están incluidos en PLENGTH.

Los pasos serían los siguientes:

1. Parsear el byte [Extended Code].
2. Parsear el byte [Code] y almacenarlo en una variable para posteriormente saber que contiene [Value].
3. Si [Code] es mayor o igual que 0x80, se parsea el siguiente byte como [VLength] para saber la longitud que tendrá [Value], sino, se sabe que la longitud será de 1 byte.
4. Se parsea el contenido de [Value], atendiendo a [Extended Code], [Code] y [VLength], que ya han sido parseados.
5. Si no han sido parseados todos los bytes de Payload, se vuelve al paso 1.

5.3.6 Paquete de ejemplo

En la siguiente tabla (Tabla 5-3) se puede observar un ejemplo de un paquete de forma detallada.

Byte	Valor	Definición
[0]	0xAA	[SYNC]
[1]	0xAA	[SYNC]
[2]	0x06	[PLENGTH] longitud de la carga útil = 6 bytes
[3]	0x02	[CODE]: POOR_SIGNAL
[4]	0x20	Valor de la señal = 32/255
[5]	0x04	[CODE]: ATTENTION
[6]	0x12	Valor del nivel de atención = 18%
[7]	0x05	[CODE]: MEDITATION
[8]	0x60	Valor del nivel de relajación = 96%
[9]	0x62	[CHKSUM] (complemento a1 inverso de 8 bits de la suma de los bytes de PAYLOAD)

Tabla 5-3 Ejemplo de paquete [27].

5.4 Funcionalidad

5.4.1 Introducción

En primer lugar, en este punto se va a describir la funcionalidad de este trabajo. Por lo tanto para empezar se va a detallar el coche teledirigido utilizado, tanto a nivel electrónico, como a nivel de funcionamiento. Una vez detallado el coche, se pasará a estudiar y explicar cómo se realiza dicha funcionalidad a través del microprocesador y con la información recibida por MindWave.

5.4.2 Coche teledirigido

5.4.2.1 Introducción

Para realizar este Proyecto se ha adquirido un coche teledirigido de juguete, que puede ser comprado en cualquier superficie comercial. El coche funciona con un mando a distancia por radiofrecuencia, es decir, las señales de control del coche se transmiten a través de dicho mando. Para realizar esta tarea ha habido que desmontar el mando del juguete y adaptarlo, como se explicará más adelante. Por otro lado el propio coche ha sido abierto para modificar su velocidad, debido a que ésta era muy alta para poder dar tiempo a controlar con la mente el movimiento. Pero esto es algo que también veremos más adelante.

5.4.2.2 Coche teledirigido

Como se comenta en la introducción a este apartado, el coche es un coche de juguete por control remoto.

Figura 5-6 Coche teledirigido.

Debido a que el coche funcionaba a una alta velocidad, se le ha insertado una resistencia a la entrada del motor, para que baje las revoluciones. Se intentó realizar una PWM para controlar el motor a través del micro, pero esto no fue posible debido a que las señales del mando no eran lo suficientemente rápidas para poder trabajar con esa funcionalidad.

Para la resistencia, se han utilizado dos resistencias de 10Ω en paralelo y dos en serie de 1Ω, formando una resistencia total de unos 7Ω. Esta resistencia es suficiente para que el coche tenga potencia de arranque y a la vez para que la velocidad no sea muy excesiva. En la siguiente figura (Figura 5-8) se puede ver el motor con el conjunto de resistencias.

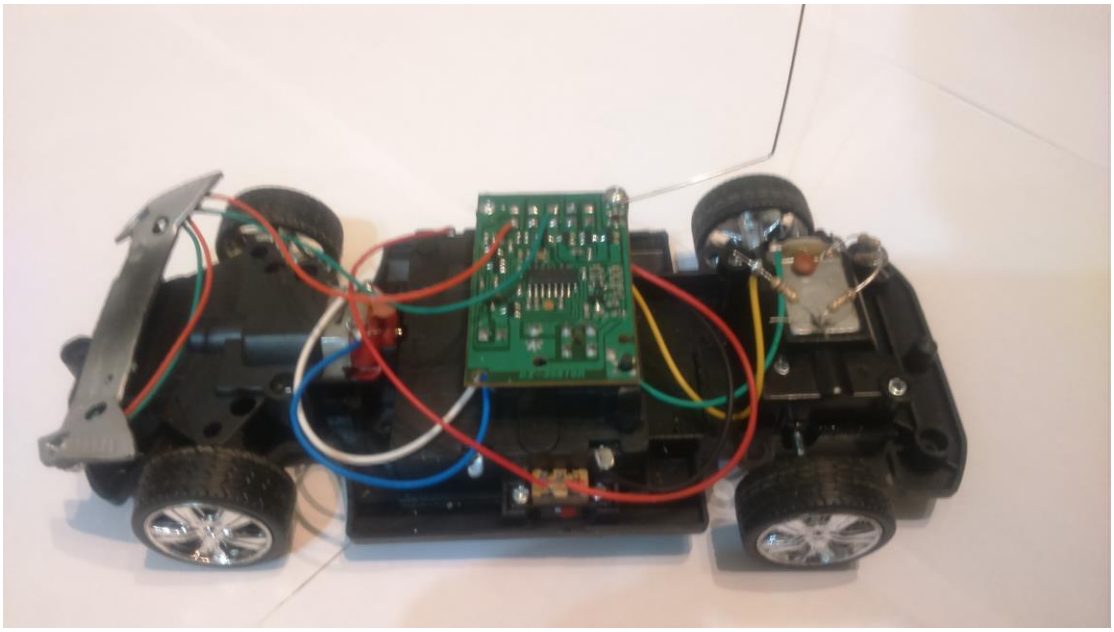

Figura 5-7 Coche desmontado.

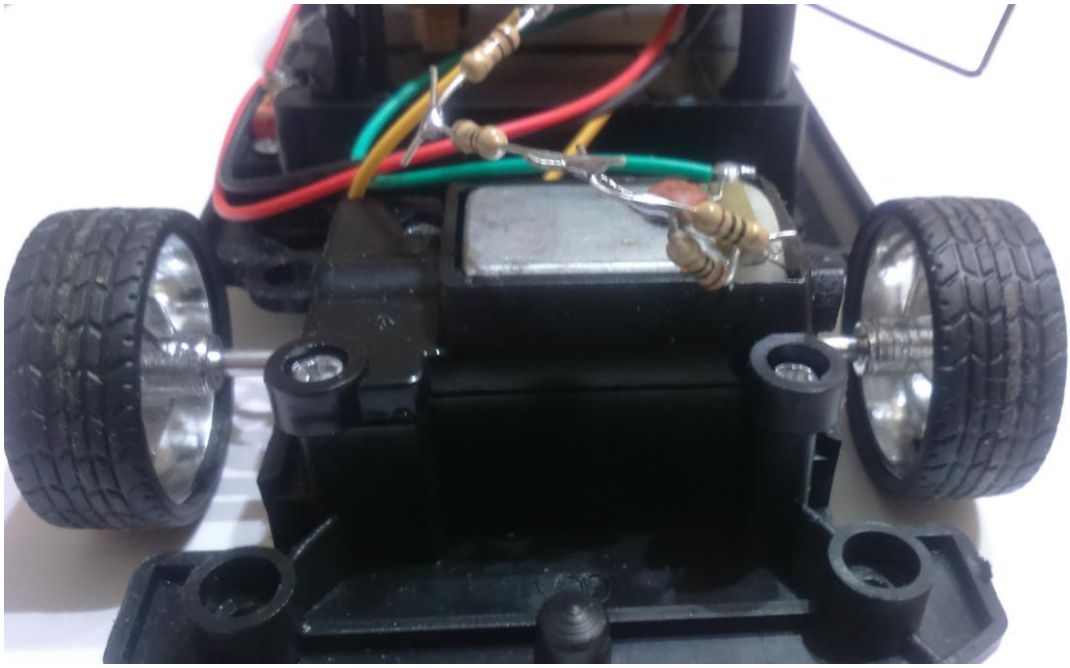

Figura 5-8 Motor con resistencias.

Para el funcionamiento del coche, se utiliza un mando a distancia que funciona por radiofrecuencia. El circuito que se implementa en dicho mando es el Tx2S [28].

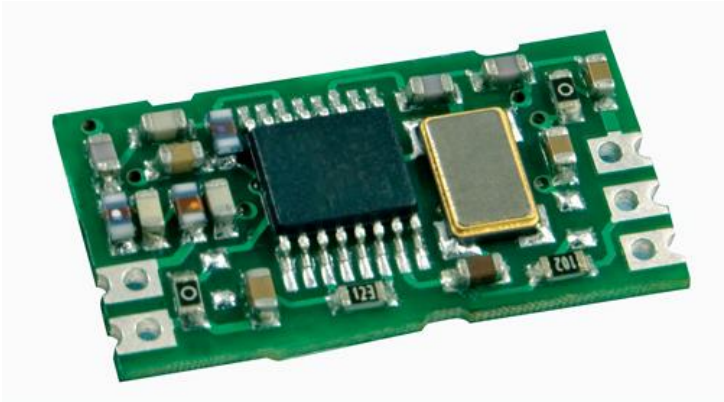

Figura 5-9 Tx2S.

Este módulo es una placa PCB en miniatura que funciona como un radio transmisor UHF [29], que implementa un enlace a 40 kbps a una distancia de unos 75 metros en interiores y 300 en exteriores. Opera entre 2.2V y 4V.

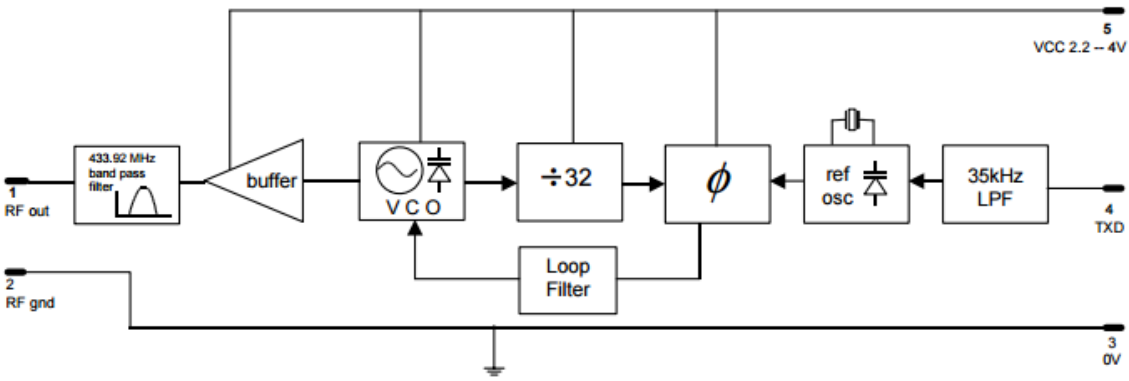

Figura 5-10 Diagrama de bloques Tx2S [28].

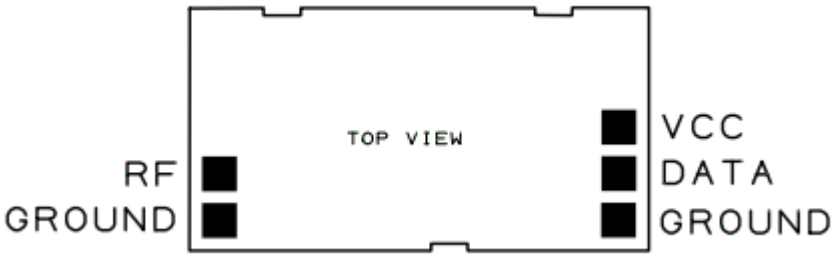

Figura 5-11 Pinout Tx2S [28].

Descripción de los pines:

- RF output: 50Ω de salida para la antena. Internamente aislado.
- RF ground: internamente conectado al pin 3 (0V).
- VCC: +2.2V a +4V de alimentación en continua (DC).
- Ground: tierra. Internamente conectada al pin 2 y 3.
- Data: entrada de la modulación. Acepta los datos digitales en serie en niveles de 0V y de 4V.

Las órdenes que se pueden mandar desde el mando a través del integrado Tx2S son:

- Ir hacia delante.
- Ir hacia atrás.
- Girar las ruedas a izquierda.
- Girar las ruedas a derecha.

Ya que los pulsadores se van a accionar desde el microprocesador, ha habido que abrir el mando para realizar una serie de modificaciones.

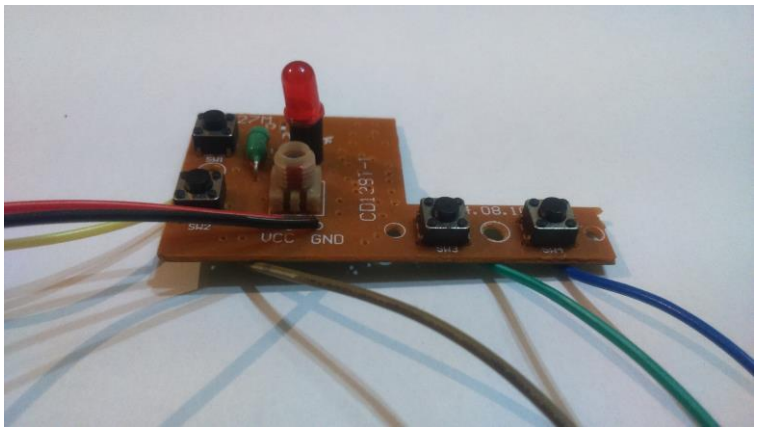

Figura 5-12 Modificaciones mando 1.

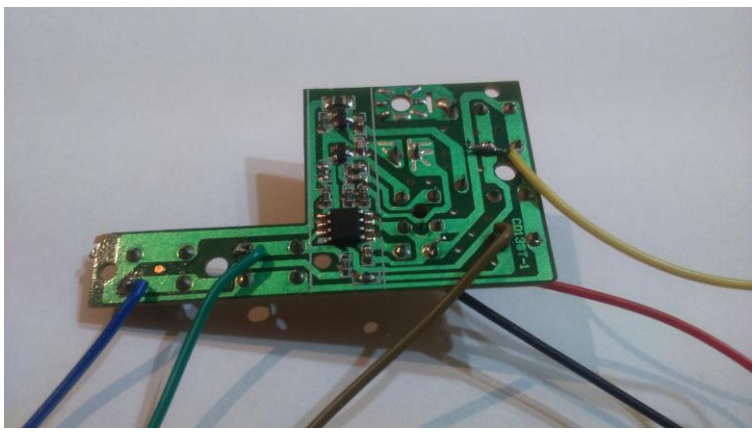

Figura 5-13 Modificaciones mando 2.

En primer lugar se han soldado los cables de alimentación y tierra a sus respectivos pines en la placa del mando. Dichos cables irán a los pines del micro correspondientes de alimentación VCC (3.3V) y tierra GND.

Los pulsadores funcionan como relés, es decir, normalmente están abiertos y al accionarlos cortocircuitan la pista exterior, dónde se encuentra la alimentación, con la pista que va al integrado, que indica la orden correspondiente. En este caso se han soldado 4 cables, cada uno a una pista según la acción, de forma que el microprocesador pone a nivel alto (3.3V) el pin que hayamos seleccionado para cada función, poniendo esos 3.3V en la entrada del integrado oportuna.

El conjunto del sistema quedaría como se aprecia en las siguientes figuras.

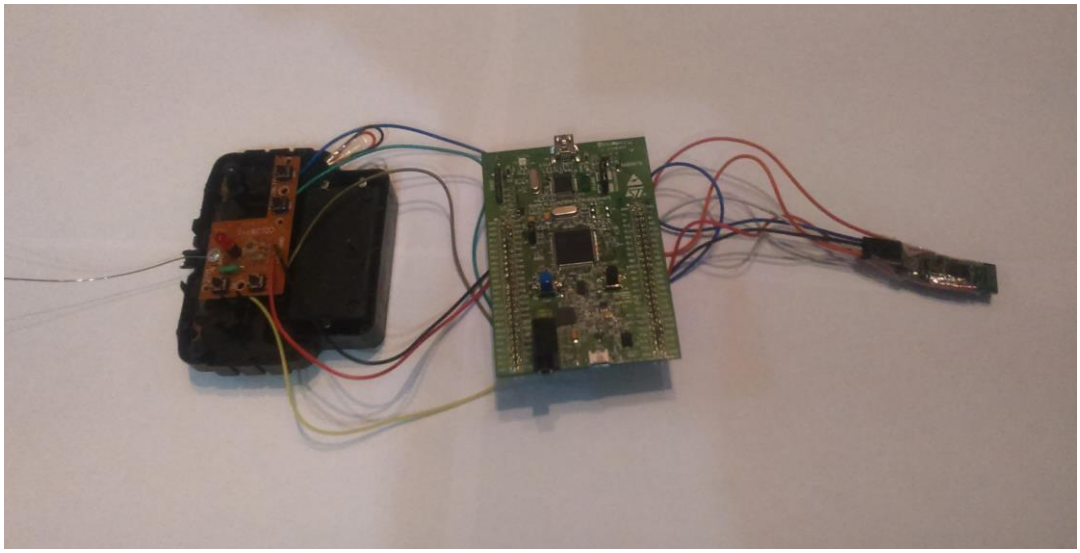

Figura 5-14 Sistema implementado.

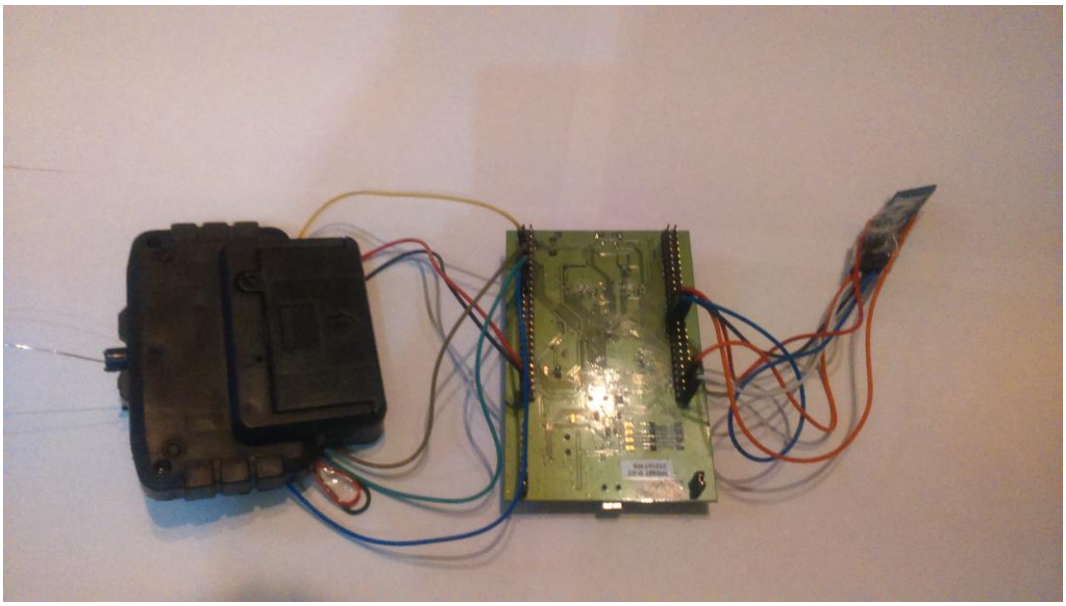

Figura 5-15 Sistema implementado boca abajo.

Primero se aprecia el módulo de bluetooth, que recepcionará las señales del casco de ondas cerebrales. Después, dichas señales van al microprocesador que realiza las operaciones necesarias y envía las órdenes al mando del coche que tiene conectado.

5.4.3 Programación de la funcionalidad

5.4.3.1 Parpadeo

Para implementar el parpadeo de forma que las ruedas giren a izquierda o derecha, se ha implementado un algoritmo de captación de picos en la onda cerebral.

Se ha podido observar el comportamiento de las ondas cerebrales en el momento en el que se produce un parpadeo, y es que se produce un pico en la amplitud.

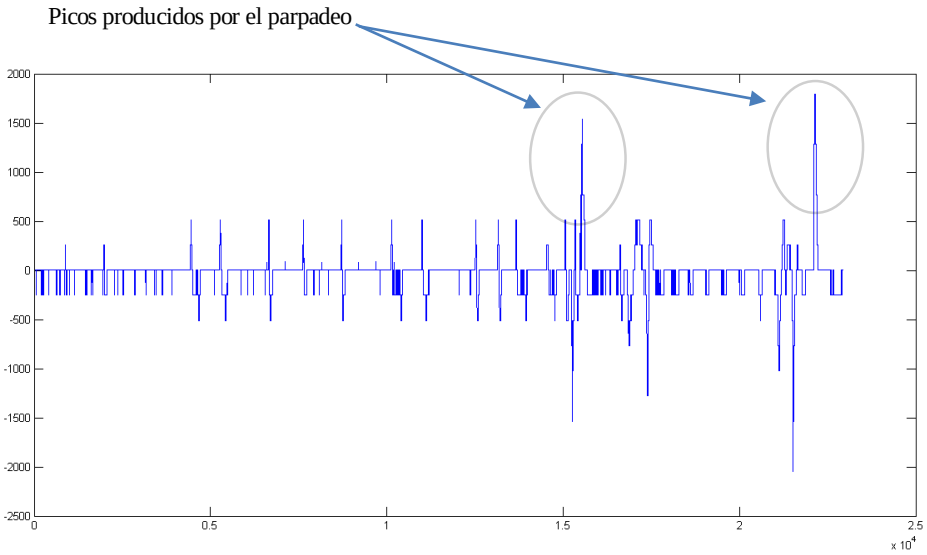

Figura 5-16 Detección parpadeo en onda cerebral.

Por lo tanto, para realizar esta operación, se van guardando las ondas cerebrales en una FIFO, actualizándola constantemente, ya que las ondas llegan 512 veces por segundo. La FIFO se ha definido con un tamaño de 450 posiciones, indexadas con una variable "j". Lo que se hace es que si la señal recibida es mayor que cero, vamos guardando los valores en la FIFO restándole la primera que entró para introducir la nueva de forma que no se desborde. A la vez, se guarda en un acumulador el valor de la suma de todos los valores de la FIFO, actualizando dicho valor a cada momento. Una vez que el buffer FIFO se ha llenado se pone a cero el índice que lo recorre de forma que el siguiente valor se introduce de nuevo en la posición cero.

```
valor_resta=FIFO[j];
FIFO[j++]=raw;
bufferSum += raw - valor_resta;
if (j==450){ j=0;}
```

Mientras se realizan estas operaciones, se calcula a tiempo real la media de los valores del buffer y se escala por un factor para observar cuando los picos producidos por el parpadeo sobrepasan dicho umbral. Cuando se producen estos picos y son mayores que el umbral marcado se considera que ha habido parpadeo.

Aun cuando se ha captado un parpadeo no se giran las ruedas. Se necesitan dos parpadeos seguidos en un tiempo de 2 segundos, para evitar posibles parpadeos erróneos, para realizar la funcionalidad de girar las ruedas. Dicha funcionalidad tiene 5 fases:

1. Primero las ruedas se encuentran en la posición inicial.
2. Tras dos parpadeos las ruedas se giran a la izquierda.
3. Con otros dos parpadeos se vuelve a la posición inicial, las ruedas se ponen rectas.
4. Si se vuelve a parpadear dos veces, ahora las ruedas giran a la derecha.
5. Finalmente con dos parpadeos más se vuelve a 1.

Para esta funcionalidad se ha hecho uso de los leds de la placa de desarrollo. Con el primer parpadeo, empieza el contador de 2 segundos y a la vez se enciende y apaga el led rojo. Cuando se ha captado el segundo parpadeo, se enciende el led verde.

5.4.3.2 Hacia delante o hacia atrás

Estas operaciones son más sencillas. Son similares en cuanto a funcionalidad, simplemente cambiando el sentido de la marcha, y la recepción de la Atención o Relajación.

Para ir hacia delante, se está muestreando constantemente los valores de Atención, de forma que si superan el 80%, se considera concentración suficiente para dar la orden. Pero hay que estar un mínimo de 2 segundos manteniendo dicho nivel de atención, para evitar posibles situaciones indeseadas. Cuando se ha estado ese tiempo concentrado, se da la orden de poner el pin, conectado a la pista del mando de ir hacia delante, a nivel alto. Durante ese tiempo en el que el nivel de atención supera el umbral del 80%, el led naranja de la placa se enciende, para tener una señal visual. Se ha programado un tiempo de unos 2 segundos con el pin activo cada vez que se da la orden, para que el coche no esté continuamente en movimiento sino que vaya realizando la trayectoria a trazos y así poder controlar la dirección lo máximo posible.

Para ir hacia detrás el comportamiento es exactamente el mismo, utilizando la Relajación en vez de la Atención. En este caso el led azul es el que se ilumina al estar con un nivel de relajación superior al 80%.

6 CONCLUSIONES

Se ha hecho una pequeño estudio acerca del cerebro humano, la técnica de la EEG (utilizada en este proyecto) y de las interfaces cerebro-ordenado (BCI). Es este campo, el que está creciendo, principalmente en aplicaciones biomédicas. Sin embargo existen muchas otras aplicaciones del sistema BCI que están comenzando a explorarse y que pronto darán su fruto. La idea básica es que pacientes o personas con movilidad reducida parcial o total puedan controlar elementos externos haciendo uso de su cerebro, con un entrenamiento previo. Se ha conseguido que un paciente pueda mover el ratón de un ordenador o jugar a videojuegos.

Bajo mi punto de pista las investigaciones de BCI's deben centrarse en métodos no invasivos, como es el EEG, que permite al paciente o individuo no correr riesgo alguno.

Sin duda, en un futuro no muy lejano, esta tecnología estará muy implantada y tal vez no solo para personas con problemas, sino dentro del entorno diario y cotidiano de todas.

Por otra parte se ha hecho un estudio del microprocesador STM32F4 de la familia ARM, con el fin de aplicar los conocimientos y competencias adquiridos a lo largo de la carrera y plasmarlos en un trabajo final. El trabajar con un sistema embebido y programarlo, es algo que se estudia en el Grado de Ingeniería de Tecnologías de Telecomunicación y que se ha aprovechado junto con la tecnología BCI para realizar la aplicación estudiada.

A su vez, se ha tocado un poco de electrónica en cuanto a nivel de circuitos y soldadura, a través del coche de juguete, para ajustarlo al nivel que se deseaba.

ANEXOS

Anexo A: Código implementado

A.1 Main.c

```
#include "stm32f4xx.h"
#include "gpio.h"
#include "rcc.h"
#include "led.h"
#include "uart_interfaz.h"
#include "uart.h"
#include "hc05.h"
#include "parsePacketPayload.h"
#include "timcont.h"
#include "coche.h"


uint32_t          temporizador=0; //Variable global que se incrementa a cada
segundo haciendo uso del timer 7
uint32_t          temporizador4=0;//Variable global que se incrementa a cada
segundo haciendo uso del timer 4

/*Función main: este fichero sirve para inicializar las partes del micro ARM
que se van a usar*/
int main (void)
{

      led_init(); //Inicializa los leds, para su posterior uso.

      coche_init(); //Inicializa los pines conectados al mando del coche,
para activarlos a nivel alto cuando corresponda.

      tim_init(); //Inicializa todos los timers usados.

      uart_interfaz_init(); //Inicializa la uart2. Se ha hecho uso de dicha
uart para depurar el código haciendo uso del hyperterminal.

      hc05_init(); //Inicializa el módulo hc05 de bluetooth, conecta el micro
con el casco y a su vez parsea los paquetes para mandarlos a analizar en otro
script.

      while(1){}

}
```

A.2 Led.c

```
#include "gpio.h"
#include "led.h"
#include "rcc.h"

/*Función led_init: inicializa los pines correspondientes a los leds*/
void led_init(void){
	RCC_ENABLE_AHB1_PERIPHERAL(GPIOD_AHB1); //Habilita el periférico
correspondiente a los leds

	GPIO_CLEAR(GPIOD,PIN12);  //Se ponen a cero los pines
	GPIO_CLEAR(GPIOD,PIN13);
	GPIO_CLEAR(GPIOD,PIN14);
	GPIO_CLEAR(GPIOD,PIN15);

	SET_GPIO_AS_OUTPUT(GPIOD, PIN12); //Se configuran los pines de los leds
como salidas
	SET_GPIO_AS_OUTPUT(GPIOD, PIN13);
	SET_GPIO_AS_OUTPUT(GPIOD, PIN14);
	SET_GPIO_AS_OUTPUT(GPIOD, PIN15);

}
/*Función led_off: sirve para apagar el led correspondiente*/
void led_off(uint8_t led){
	switch (led)
	{
		case 1:
			GPIO_CLEAR(GPIOD, PIN12);
			break;
		case 2:
			GPIO_CLEAR(GPIOD, PIN13);
		  break;
		case 3:
			GPIO_CLEAR(GPIOD, PIN14);
			break;
		case 4:
			GPIO_CLEAR(GPIOD, PIN15);
			break;
	}
}
/*Función led_on: sirve para encender el led correspondiente*/
void led_on(uint8_t led){
	switch (led)
	{
		case 1:
			GPIO_SET(GPIOD, PIN12);
			break;
		case 2:
			GPIO_SET(GPIOD, PIN13);
		  break;
		case 3:
			GPIO_SET(GPIOD, PIN14);
			break;
		case 4:
			GPIO_SET(GPIOD, PIN15);
			break;
	}
}
/*Función led_toggle: sirve para encender/apagar el led correspondiente*/
```

```
void led_toggle(uint8_t led){
	switch (led)
	{
		case 1:
			GPIO_TOGGLE(GPIOD, PIN12);
			break;
	  case 2:
			GPIO_TOGGLE(GPIOD, PIN13);
		  break;
		case 3:
			GPIO_TOGGLE(GPIOD, PIN14);
			break;
		case 4:
			GPIO_TOGGLE(GPIOD, PIN15);
			break;
	}
}
```

A.3 Led.h

```
#ifndef LED_H
#define LED_H

#include "stm32f4xx.h"

#define led_green_on()    led_on(1)
#define led_orange_on()   led_on(2)
#define led_red_on()      led_on(3)
#define led_blue_on()     led_on(4)

#define led_green_off()   led_off(1)
#define led_orange_off()  led_off(2)
#define led_red_off()     led_off(3)
#define led_blue_off()    led_off(4)

#define led_green_toggle()   led_toggle(1)
#define led_orange_toggle()  led_toggle(2)
#define led_red_toggle()     led_toggle(3)
#define led_blue_toggle()    led_toggle(4)


void led_init(void);
void led_on(uint8_t);
void led_off(uint8_t);
void led_toggle(uint8_t);

#endif
```

A.4 Coche.c

```
#include "gpio.h"
#include "coche.h"
#include "rcc.h"

/*Función coche_init: sirve para habilitar e inicializar los pines conectados
al mando del coche teledirigido*/
void coche_init(void){

	RCC_ENABLE_AHB1_PERIPHERAL(GPIOD_AHB1); //Se habilitan los puertos
usados
	RCC_ENABLE_AHB1_PERIPHERAL(GPIOE_AHB1);
	RCC_ENABLE_AHB1_PERIPHERAL(GPIOB_AHB1);

	GPIO_CLEAR(GPIOD,PIN11);  //Se ponen a cero los pines
	GPIO_CLEAR(GPIOB,PIN15);
	GPIO_CLEAR(GPIOB,PIN11);
	GPIO_CLEAR(GPIOE,PIN13);

	SET_GPIO_AS_OUTPUT(GPIOD, PIN11); //Se inicializan como salidas
	SET_GPIO_AS_OUTPUT(GPIOB, PIN15);
	SET_GPIO_AS_OUTPUT(GPIOB, PIN11);
	SET_GPIO_AS_OUTPUT(GPIOE, PIN13);

}

/*Función coche_on: sirve para que el coche avance hacia delante. Se pone a
nivel alto el pin correspondiente al botón del mando que manda dicha orden */
void coche_on(void){
		GPIO_SET(GPIOD,PIN11);
}
/*Función coche_off: lo inverso a coche_on. Se pone el pin a nivel bajo para
parar el coche*/
void coche_off(void){
		GPIO_CLEAR(GPIOD,PIN11);
}
/*Función coche_atras_on: sirve para que el coche avance hacia atrás. Se pone
a nivel alto el pin correspondiente al botón del mando que manda dicha orden
*/
void coche_atras_on(void){
		SET_GPIO_AS_INPUT(GPIOD,PIN11);
		GPIO_SET(GPIOB,PIN15);
}
/*Función coche_atras_off: lo inverso a coche_atras_on. Se pone el pin a
nivel bajo para parar el coche*/
void coche_atras_off(void){
		GPIO_CLEAR(GPIOB,PIN15);
		SET_GPIO_AS_OUTPUT(GPIOD, PIN11);
}
/*Función coche_izq_on: sirve para girar las ruedas del coche a la
izquierda*/
void coche_izq_on(void){
		GPIO_SET(GPIOB, PIN11);
}
/*Función coche_izq_off: pone a nivel bajo el pin correspondiente de forma
que las ruedas vuelven a su posición inicial*/
void coche_izq_off(void){
		GPIO_CLEAR(GPIOB, PIN11);
}
```

```
/*Función coche_der_on: sirve para girar las ruedas del coche a la derecha*/
void coche_der_on(void){
                  SET_GPIO_AS_INPUT(GPIOB,PIN11);
                  GPIO_SET(GPIOE,PIN13);
}
/*Función coche_der_off: pone a nivel bajo el pin correspondiente de forma
que las ruedas vuelven a su posición inicial*/
void coche_der_off(void){
                  GPIO_CLEAR(GPIOE,PIN13);
                  SET_GPIO_AS_OUTPUT(GPIOB, PIN11);
}
```

A.5 Coche.h

```
#ifndef COCHE_H
#define COCHE_H

#include "stm32f4xx.h"

void coche_init(void);
void coche_on(void);
void coche_off(void);
void coche_izq_on(void);
void coche_izq_off(void);
void coche_der_on(void);
void coche_der_off(void);
void coche_atras_on(void);
void coche_atras_off(void);

#endif
```

A.6 Timcont.c

```
#include "stm32f4xx.h"
#include "timer.h"
#include "timcont.h"
#include "rcc.h"
#include "print.h"
#include "led.h"

/*Función tim_init: sirve para habilitar e inicializar los timer 7 y 4 para
su posterior uso*/
void tim_init(void){

      RCC_ENABLE_APB1_PERIPHERAL(TIM7_APB1); //Se habilitan los perifericos
correspondientes
      RCC_ENABLE_APB1_PERIPHERAL(TIM4_APB1);
      RCC_ENABLE_APB1_PERIPHERAL(TIM6_APB1);

      TIM7->CR1=0x84;//Se pone un 84 en el registro CR1, para configuar el
timer de forma que siga contando con cada interrupción y que dicha
interrupción sea cada x tiempo, especificado en el registro ARR
```

```
	TIM7->ARR=0x3E7;//Número de veces que tiene que pasar para que salte
una interrupción = 1 segundo
	TIM7->DIER=0x1;//Se pone un 1 en el registro DIER para habilitar la
interrupción

	TIM4->CR1=0x84;//Igual que timer7
	TIM4->ARR=0x3E7;
	TIM4->DIER=0x1;

	NVIC_SetPriority(TIM7_IRQn,1); // prioridad de la interrupción
	NVIC_EnableIRQ(TIM7_IRQn); // se habilita la interrupción del timer 7
por overflow
	SET_TIMER_PRESCALER(TIM7,(uint16_t)(((SystemCoreClock/1000))-1)); // Se
ha configurado el timer con este preescaler.

	NVIC_SetPriority(TIM4_IRQn,1);		// igual que timer 7
	NVIC_EnableIRQ(TIM4_IRQn);
	SET_TIMER_PRESCALER(TIM4,(uint16_t)(((SystemCoreClock/1000))-1));

	SET_TIMER_PRESCALER(TIM6,(uint16_t)(((SystemCoreClock/1000000))-1)); //
1 MHZ Timer. Este timer se usa para las funciones delay_*
	UPDATE_TIMER_REGISTER(TIM6);
	ENABLE_TIMER(TIM6);

}
/*Las funciones tim_on(),tim_off() y tim_deinit, sirven para iniciar el timer
7, pararlo o deshabilitarlo*/
void tim_on(){
	UPDATE_TIMER_REGISTER(TIM7);
	ENABLE_TIMER(TIM7);
}

void tim_deinit(void){
	DISABLE_TIMER(TIM7);
	RCC_DISABLE_APB1_PERIPHERAL(TIM7_APB1);
}
void tim_off(void){
	DISABLE_TIMER(TIM7);
}
/*Las funciones tim4_on(), tim4_off() y tim4_deinit(), son iguales que las
del timer 7 pero para el timer 4*/
void tim4_on(){
	UPDATE_TIMER_REGISTER(TIM4);
	ENABLE_TIMER(TIM4);
}

void tim4_deinit(void){
	DISABLE_TIMER(TIM4);
	RCC_DISABLE_APB1_PERIPHERAL(TIM4_APB1);
}
void tim4_off(void){
	DISABLE_TIMER(TIM4);
}

extern uint32_t		temporizador;
extern uint32_t		temporizador4;
```

```
void TIM7_IRQHandler(void)
{
//Todo lo que escribamos aquí se ejecutará cuando salte la interrupción del
timer 7

led_red_toggle();
SET_TIMER_OV_INTERRUPT_ENABLE(TIM7);//Hay que asegurarse de borrar el flag

temporizador++; //Se incrementa la variable temporizador cada segundo
}

void TIM4_IRQHandler(void)
{
//Todo lo que escribamos aquí se ejecutará cuando salte la interrupción del
timer 4

SET_TIMER_OV_INTERRUPT_ENABLE(TIM4);//Hay que asegurarse de borrar el flag

temporizador4++; //Se incrementa la variable temporizador4 cada segundo
}

/*Las funciones delay_*() sirven para provocar un retraso parando la
ejecución del programa el tiempo deseado*/
void delay_deinit(void){
      DISABLE_TIMER(TIM6);
      RCC_DISABLE_APB1_PERIPHERAL(TIM6_APB1);
}

void delay_us(uint16_t TIME_US){

      uint16_t time;
      time=TIME_US; //time compensation
      TIM6->CNT=0;
  while((uint16_t)(TIM6->CNT) <= time);
}

void delay_sec(uint8_t TIME_SEC){

      uint16_t i;

      i=TIME_SEC;

      while (i)
      {
            delay_ms(1000);
            i--;
      }


}
void delay_ms(uint16_t TIME_MS){

      uint16_t i;

      i=TIME_MS;

      while (i)
      {
            delay_us(1000);
            i--;
      }
}
```

A.7 Timcont.h

```
#ifndef TIMCONT_H
#define TIMCONT_H

#include "stm32f4xx.h"

void tim_init(void);
void tim_deinit(void);
void TIM7_IRQHandler(void);
void tim_on(void);
void tim_off(void);
void tim4_deinit(void);
void TIM4_IRQHandler(void);
void tim4_on(void);
void tim4_off(void);

void delay_deinit(void);
void delay_us(uint16_t);
void delay_ms(uint16_t);
void delay_sec(uint8_t);

#endif
```

A.8 Uart_interfaz.c

```
#include "stm32f4xx.h"
#include "gpio.h"
#include "rcc.h"
#include "uart.h"
#include "uart_interfaz.h"
#include "led.h"

/*Función uart_interfaz_init: sirve para inicializar la uart2*/
uint8_t uart_interfaz_init(void)
{
      uint8_t error=0;

      //////////////
      // USART 2 //
      //////////////
  RCC_ENABLE_AHB1_PERIPHERAL(GPIOA_AHB1);// se habilita el periferico
correspondiente

  SET_GPIO_AS_AF(GPIOA,PIN2,AF7);
  SET_GPIO_AS_AF(GPIOA,PIN3,AF7);

  RCC_ENABLE_APB1_PERIPHERAL(USART2_APB1);  // Enable USART2 clock
  UART_ENABLE(USART2,115200,UART_TX_RX); // Se configura la USART2 a 115200
baudios
  NVIC_SetPriority(USART2_IRQn,1);          // prioridad
  NVIC_EnableIRQ(USART2_IRQn);
  UART2_SET_RX_FUNCTION(uart_interfaz_on_rx_byte);
```

```
  return error;
}
/*Función uart_interfaz_on_rx_byte: sirve para mandar lo que haya en Buf por
la uart al hyperterminal en este caso*/
void uart_interfaz_on_rx_byte(uint8_t Buf){
      UART2_SEND(Buf);
}
```

A.9 Uart_interfaz.h

```
#ifndef HOST_INTERFACE_H
#define HOST_INTERFACE_H

#include "stm32f4xx.h"

uint8_t uart_interfaz_init (void);
void uart_interfaz_on_rx_byte(uint8_t);


#endif
```

A.10 HC05.c

```
#include "stm32f4xx.h"
#include "hexutil.h"
#include "uart.h"
#include "rcc.h"
#include "gpio.h"
#include "hc05.h"
#include "print.h"
#include "parsePacketPayload.h"
#include "timcont.h"


#define HC05_OK      0
#define HC05_TIMEOUT 1
#define TIME_OUT      0x7FFF


#define KEY_UP      GPIO_SET(GPIOB,PIN4)
#define KEY_DOWN    GPIO_CLEAR(GPIOB,PIN4)

#define  RESET_UP      GPIO_SET(GPIOB,PIN5)
#define  RESET_DOWN    GPIO_CLEAR(GPIOB,PIN5)

#define PARSER_SYNC_BYTE     0xAA  /* Byte de sincronización */


void  hc05_parseOK(uint8_t);
void  hc05_parser(uint8_t Buf);
void  hc05_reset(void);
```

```
void  hc05_AT_mode(void);
void  hc05_COM_mode(void);


/*Función hc05_reset: sirve para resetear el modulo bluetooth*/
void  hc05_reset(void){
      RESET_DOWN;
      delay_sec(2);
      RESET_UP;
}
/*Función hc05_AT_mode: esta función sirve para mandar los comandos AT
necesarios para configuar el modulo hc05*/
void  hc05_AT_mode(void){
      KEY_UP;
      hc05_reset();
      delay_ms(1500);
      UART_ENABLE(USART1,38400,UART_TX_RX); //Se habilita la USART1 a 38400
baudios para mandar los comandos AT necesarios para configuar el módulo hc05
y conectarlo con el casco
      NVIC_SetPriority(USART1_IRQn,3); //prioridad
      UART1_SET_RX_FUNCTION(hc05_parseOK);
}
/*Función hc05_COM_mode: sirve para conectarse al casco configurando la
velocidad de tx y rx a 5760*/
void  hc05_COM_mode(void){
      KEY_DOWN;
      hc05_reset();
      delay_ms(1500);
      UART_ENABLE(USART1,57600,UART_TX_RX);//Se habilita la USART1 a 57600
para comunicarse con el casco a esa tasa
      NVIC_SetPriority(USART1_IRQn,3);
      UART1_SET_RX_FUNCTION(hc05_parser);
}



      volatile uint8_t OK_flag;

      /*Función waitOK: sirve para ver si hay algún problema o error con el
modulo bluetooth*/
      uint8_t waitOK (void){
      uint32_t timeOut=0;
      uint8_t error=HC05_OK;

      OK_flag=0;
      while(!OK_flag)
      {
            if ((timeOut)++==TIME_OUT){error=HC05_TIMEOUT; break;}
      }

            while(!OK_flag)
      {
            if ((timeOut)++==TIME_OUT){error=HC05_TIMEOUT; break;}
      }

      return error;
}

   extern unsigned char   plength; //variable que guarda el tamaño de la
trama recibida en el paquete enviado por el casco
   extern unsigned char   payloadBytesReceived; //índice que recorre el
vector payload
```

```
   extern unsigned char   payload[256]; //vector o tabla donde se encuentran
los bytes recibidos por el casco
   extern unsigned char   payloadSum; //variable dónde se guarda la suma de
los bytes parseados para su posterior comprobación
   extern unsigned char   chksum; //variable dónde se almacena el valor del
checksum enviado en el paquete para comprobar que coincide con payloadSum

//*Función hc05_parser: esta función se encarga del parseo de los paquetes
recibidos.Funciona como una máquina de estados*/
void hc05_parser(uint8_t Buf){


      static enum RSM {SYNC, SYNC2, PLENGTH, DATA_PAYLOAD, CHKSUM}
RSM_STATE=SYNC;

      switch (RSM_STATE)
            {
            case SYNC: //1. Se comprueba que el primer byte es 0xAA, que es el
byte de sincronización que debe llegar
            if (Buf == PARSER_SYNC_BYTE){
                  RSM_STATE=SYNC2;
            }
            else{
                  RSM_STATE=SYNC;
            }
            break;

            case SYNC2://2. Se vuelve a comprobar que llega 0xAA, ya que son
necesarios dos bytes iguales para saber que lo que llega es un paquete
            if (Buf == PARSER_SYNC_BYTE){
                  RSM_STATE=PLENGTH;
            }
            else{
                  RSM_STATE=SYNC;
            }
            break;

            case PLENGTH: //3. El siguiente byte es la longitud de la trama
que se encuentra dentro del paquete
            if(Buf==PARSER_SYNC_BYTE){//si el byte que llega es igual a 0xAA,
se vuelve a comprobar
                  RSM_STATE=PLENGTH;
            }
            else if(Buf>170){//si el byte de longitud es mayor que 170 se
descarta el paquete
                  RSM_STATE=SYNC;
                  //ERROR: PLENGTH_TOO_LARGE
            }
            else{ //de lo contrario se almacena en plength el tamaño de la
trama
                  plength=Buf;
                  payloadBytesReceived = 0;
                  payloadSum = 0;
                  RSM_STATE=DATA_PAYLOAD;
            }
            break;

            case DATA_PAYLOAD: //4. Se guarda en payload la trama y se
actualiza payloadSum

                  payload[payloadBytesReceived++] = Buf;
                  payloadSum = (unsigned char)(payloadSum + Buf);
```

```
            if(payloadBytesReceived >= plength ) {//cuando se ha terminado de
guardar la trama se pasa al último estado
            RSM_STATE = CHKSUM;
            }
            break;

            case CHKSUM: //5. Se comprueba que el checksum, que es el último
byte enviado en un paquete, coincide con payloadSum

                  chksum = Buf;
                  RSM_STATE = SYNC;

            if(chksum != ((~payloadSum)&0xFF) ) {
                  //ERROR: CHKSUM HA FALLADO
            } else {// si coincide se pasa a realizar el parseo de la trama

                  //Paquete Recibido Ok
                  parsePacketPayload();
            }
            break;

      }
}


/*Función hc05_parseOK: simplemente sirve para comprobar que los comandos AT
reciben su Ok correspondiente para comprobar que todo va bien*/
void hc05_parseOK(uint8_t Buf){
      static enum OK_PARSER {OK_IDLE, O , K, R} OK_PARSER=OK_IDLE;

      switch(OK_PARSER)
      {
            case OK_IDLE:
                  if (Buf=='O')OK_PARSER=O;
            break;

            case O:
                  if (Buf=='K')OK_PARSER=K;
                  else OK_PARSER=OK_IDLE;
            break;

            case K:
                  if (Buf=='\r')OK_PARSER=R;
                  else OK_PARSER=OK_IDLE;
            break;

            case R:
                  if (Buf=='\n')OK_flag=1;
              OK_PARSER=OK_IDLE;
            break;

      }

}


/*Función hc05_init: sirve para inicializar el modulo bluetooth y conectarse
al casco*/
uint8_t hc05_init (void){

      uint8_t error;
```

```
	RCC_ENABLE_AHB1_PERIPHERAL(GPIOB_AHB1);// se habilita el periferico
correspondiente

	SET_GPIO_AS_AF(GPIOB,PIN6,AF7); //Pin6->TX
	SET_GPIO_AS_AF(GPIOB,PIN7,AF7);     //Pin7->RX

	RCC_ENABLE_APB2_PERIPHERAL(USART1_APB2);  // Enable USART1 clock
	UART_ENABLE(USART1,38400,UART_TX_RX);
	NVIC_SetPriority(USART1_IRQn,2);          //prioridad
	NVIC_EnableIRQ(USART1_IRQn);

	//Configuración de Key & Reset
	SET_GPIO_AS_OUTPUT(GPIOB,PIN4);
	SET_GPIO_AS_OUTPUT(GPIOB,PIN5);

	//Init commands
	hc05_AT_mode();//comienza el modo AT

	hc05_send ((uint8_t *)"AT+PSWD=0000\r\n", 14);//se configura la
contreseña del modulo para que coincida con la del casco. Sería: "0000"
	error=waitOK();

	hc05_send ((uint8_t *)"AT+ROLE=1\r\n", 11);//se configura el modulo
como maestro. Valor=1
	error=waitOK();

	hc05_send ((uint8_t *)"AT+CMODE=0\r\n", 12);//se configura CMODE=0 ya
que vamos a conectarnos a una dirección en concreto
	error=waitOK();

	hc05_send ((uint8_t *)"AT+UART=57600,1,0\r\n", 19);//se configura la
tasa de transmision a 57600 baudios
	error=waitOK();

	hc05_send ((uint8_t *)"AT+BIND=2068,9d,4c14f0\r\n", 24); //se fija la
dirección del casco al que nos vamos a conectar
	error=waitOK();

	hc05_COM_mode();//se pasa al modo comunicación


	return error;
}

/*Función hc05_send: esta función sirve para mandar por la UART1 los comandos
AT*/
void hc05_send (uint8_t *Buf, uint16_t Size){

  int16_t i=0;

  for (i=0;i<Size;i++)
  {
    UART1_SEND(Buf[i]);
  }

}
```

A.11 HC05.h

```
#ifndef HC05_H
#define HC05_H

#include "stm32f4xx.h"

uint8_t hc05_init(void);
void hc05_send (uint8_t *, uint16_t );

#endif
```

A.12 parsePacketPayload.c

```
#include "stm32f4xx.h"
#include "gpio.h"
#include "rcc.h"
#include "hexutil.h"
#include "uart.h"
#include "hc05.h"
#include "print.h"
#include "parsePacketPayload.h"
#include "led.h"
#include "timcont.h"
#include "coche.h"

#define PARSER_EXCODE_BYTE   0x55  /* EXtended CODE byte */

/* Definición de los códigos */
#define PARSER_CODE_POOR_QUALITY       0x02
#define PARSER_CODE_ATTENTION          0x04
#define PARSER_CODE_MEDITATION         0x05


/*---VARIBALES PARA EL PARSEO DEL PAQUETE---*/

unsigned char   plength;           //variable que guarda el tamaño de la
trama recibida en el paquete enviado por el casco
unsigned char   payloadBytesReceived;//índice que recorre el vector payload
unsigned char   payload[256]; //vector o tabla donde se encuentran los bytes
recibidos por el casco
unsigned char   payloadSum;        //variable dónde se guarda la suma de los
bytes parseados para su posterior comprobación
unsigned char   chksum;            //variable dónde se almacena el valor del
checksum enviado en el paquete para comprobar que coincide con payloadSum

/*-----------------------------------------*/


/*---VARIABLES PARA BLINK---*/

int16_t           raw;                  //variable dónde se almacena la raw-
wave (onda cerebral)
```

```
uint16_t          rawvalue[2];//vector para almacenar los dos bytes que forman
raw

int16_t FIFO[450];                  //vector que se usa como buffer circular
para calcular la media de la señal raw.
uint16_t j=0;                       //índice que recorre el vector FIFO
signed long bufferSum=0;      //acumulador
unsigned char     flag=0;           //variable que se usa para entrar en una
condición "if" una vez se ha detectado el primer pico
int16_t           valor_resta;//variable usada para almacenar la primera
muestra introducida en FIFO para luego sacarla del buffer y calcular la media
a tiempo real
signed long       media=0;    //media de las 450 muestras del buffer
double                  escala = 8; //valor usado para escalar la media para
considerar si se producen picos debidos al parpadeo
signed long       media_escalada=0;//variable dónde se almacena la media
escalada

unsigned char Nblink=0;       //variable usada para contar los picos de raw-
signal y verificar si se ha producido un parpadeo
unsigned char contador=0;     //variable usada para realizar una cuenta de 100
muestras a partir de la primera recibida como posible parpadeo

uint8_t                 blink_flag=0;//variable que se usa para detectar un
segundo parpadeo

/*-------------------------*/

/*--VARIABLES DE LOS TEMPORIZADORES--*/
extern uint32_t         temporizador;
extern uint32_t         temporizador4;
/*-----------------------------------*/

/*--VARIABLES PARA EL PARSEO DE LA TRAMA*/

unsigned char value_att;            //variable que contiene el valor de
atención
unsigned char value_med;            //variable que contiene el valor de
meditación
unsigned char value_signal;         //variable que contiene el valor de la
fuerza de la señal

uint32_t          tempor_att=0;     //variable usada para contar hasta 2
segundos de atención
uint32_t          tempor_med=0;     //variable usada para contar hasta 3
segundos de meditación
uint32_t          alante_tim=0;     //variable que se usa para realizar una
cuenta de 2 segundos cuando el coche va hacia delante
uint32_t          atras_tim=0;      //variable que se usa para realizar una
cuenta de 2 segundos cuando el coche va hacia atrás

unsigned char alante=0;             //bandera para ordenar que el coche vaya
hacia delante
unsigned char atras=0;              //bandera para ordenar que el coche vaya
hacia atras
unsigned char flag_att=0;           //bandera para iniciar la cuenta de
atención
unsigned char flag_med=0;           //bandera para iniciar la cuenta de
meditación
```

```
uint8_t empieza=0;                          //bandera usada para comenzar el
funcionamiento una vez han pasado 14 segundos, usados para estabilizar la
señal
uint8_t firstime=0;                         //bandera para comenzar la cuenta de
los 14 segundos de establecimiento de la señal

unsigned char izq_der=2;            //variable usada para la fucionalidad del
parpadeo (ruedas a izquierda o derecha)

/*-------------------------------------*/


uint8_t parsePacketPayload(void)
 {

    unsigned char i = 0;                        //índice que recorre la trama
    unsigned char extendedCodeLevel = 0;//variable que se podría usar en caso
de que el casco trabajara con más códigos (más tipos de datos)
    unsigned char code = 0;                     //variable que incluye el
código dónde se informa de que información se va a procesar
    unsigned char numBytes = 0;                 //número de bytes que se están
parseando

    /* Se parsean todos los bytes de payload[] */
    while( i < plength ) {

        /* Se parsea un posible EXtended CODE bytes */
        while( payload[i] == PARSER_EXCODE_BYTE ) {
            extendedCodeLevel++;
            i++;
        }

        /* Se parsea el CODE (código), almacenando su valor en code*/
        code = payload[i++];

        /* Se parsea el valor de la longitud */
        if( code == 0x80 ) numBytes = 2;
            else if (code>0x80) numBytes = payload[i++];
        else                numBytes = 1;

        if(firstime==0){// la primera vez se inicializa el timer 7 para
contar 14 segundos mientras se estabiliza la señal
                tim_on();
                firstime=1;
            }

            if(firstime==1 && temporizador==14){//cuando han pasado los 14
segundos se activa la bandera empieza para que se empiecen a parsear las
tramas
                empieza=1;
              tim_off();
                temporizador=0;
                tim4_on(); //se pone en marcha el timer 4, que será usado
mas adelante
            }

            if(alante==1 && atrás!=1 && empieza==1){//Funcionalidad: el coche
va hacia delante
                if(   alante_tim>(temporizador4-2)){
                      coche_on();                                   //durante 2
segundos coche_on()
                }else{
```

```
                        alante=0;
                        alante_tim=0;
                        coche_off();
                  }
            }

            if(atras==1 && adelante!=1 && empieza==1){//Funcionalidad: el
coche va hacia atras
                  if(   atras_tim>(temporizador4-2)){
                        coche_atras_on();                  //durante 2
segundos coche_atras_on()
                  }else{
                        atras=0;
                        atras_tim=0;
                        coche_atras_off();
                  }
            }

            if(izq_der==2 && empieza==1){             //Funcionalidad: giro de
las ruedas
                  coche_izq_off();                        //1. Las ruedas en
posición inicial
                  coche_der_off();
            }else if(izq_der==3 && empieza==1){       //2. Tras dos parpadeos,
las ruedas giran a la izquierda
                  coche_izq_on();


            }else if(izq_der==0 && empieza==1){       //3. Tras dos parpadeos,
las ruedas vuelven a la posición inicial
                  coche_izq_off();
                  coche_der_off();

            }else if(izq_der==1 && empieza==1){       //4. Tras dos parpadeos,
las ruedas giran a la derecha y al parpadear dos veces se vuelve a 1.
                  coche_der_on();

            }


            if(extendedCodeLevel==0){           //En el caso de códigos
cortos, como es el caso se parsea la trama

                  switch( code ) {

                  /* [CODE]: POOR_SIGNAL */
                  case( 0x02 ):

                  value_signal=payload[i];

                 if(value_signal>180){
                        //ERROR: MAL CONTACTO - SEÑAL DEBIL
                        print((uint8_t *)"Mal contacto\n\r",14,0);
                        print((uint8_t *)"\n\r",2,0);
                  }

                  break;

                  /* [CODE]: RAW_SIGNAL */

                  case( 0x80 ):
```

```
                    rawvalue[1]=payload[i++]; //Se guardan los dos bytes que
forman la señal raw y se ordenan para tener la señal en una sola variable:
raw.
                    rawvalue[0]=payload[i];
                    raw = (int16_t)((rawvalue[0]<<8) | rawvalue[1]); //valor de
la raw-wave

                    if(raw>0){
                          valor_resta=FIFO[j];                //Cuando la señal
es positiva se almacena su valor en un buffer FIFO

                          FIFO[j++]=raw;
                          bufferSum += raw - valor_resta;//Se guarda en un
acumulador el valor de la suma de todas las muestras del buffer menos la
primera
                          if(j==450){
                                j =0;                              //se pone a 0
el índice cuando se llega al final del buffer

                          }
                          media = bufferSum/450;           //se calcula la media
de las 450 muestras
                          media_escalada=media*escala;    //se escala la media
para comprobar si hay parpadeo posteriormente
                    }

                    if(raw>0 && raw>=media_escalada && contador<100 &&
empieza==1){
                          Nblink++;                                    //si la
señal es mayor que la media escalada se considera un posible parpadeo
                          flag=1;                                            //se
cuentan hasta 100 muestras más, incrementando Nblink cada vez que raw sea
mayor que la media escalada
                    }

                    if(flag==1){
                          contador++;                                  //se
incrementa contador a partir de la primera muestra mayor que la media
escalada hasta llegar a 100
                    }

                    if(contador==100 && blink_flag==0 && empieza==1){
                          flag=0;
                          contador=0;

                          if(Nblink>=32){               //cuando se ha llegado a
100 se ponen a cero las variables correspondientes y se comprueba si Nblink
es mayor que 31
                                blink_flag=1;           //en caso de que sea
mayor, se considera parpadeo y se pone en marcha el timer 7 como contador
(parpadeando mientras el led rojo)
                                Nblink=0;
                                tim_on();
                                temporizador=0;
                          }else{
                                Nblink=0;
                                blink_flag=0;
                                tim_off();
                                led_red_off();
                          }
                    }else if(contador==100 && blink_flag==1 && empieza==1){
```

```
                        flag=0;
      //se realiza la misma función anteriormente descrita para detectar un
segundo parpadeo en 2 segundos
                        contador=0;

                        if(Nblink>=30 && temporizador<=2){ //si en 2 segundos
se ha detectado el parpadeo, quiere decir que ha habidos 2 parpadeos
correctos
                              Nblink=0;                              //y se
realiza el giro de ruedas correspondiente mendiente el incremento de la
variable izq_der
                              led_green_toggle();
                              temporizador=0;
                              tim_off();
                              led_red_off();
                              blink_flag=0;
                              izq_der++;
                              izq_der&=3;
                        }
                        if(temporizador>2){
                              blink_flag=0;
                              temporizador=0;
                              tim_off();
                              led_red_off();
                              Nblink=0;
                        }else{
                              Nblink=0;
                        }
                  }

                  break;

                  /* [CODE]: ATTENTION_SENSE */
            case( 0x04 ):

                  value_att=payload[i];                    //se almacena en
value_att el valor de la atención

                  if(value_att>76 && empieza==1){          //si es mayor que
75% se inicia una cuenta para mover el coche hacia delante si se está
concentrado
                        led_orange_on();                   //durante 2
segundos seguidos

                        if(flag_att==0){
                              tempor_att=temporizador4;
                              flag_att=1;
                        }
                        if(alante_tim<(temporizador4-3)){
                              alante=0;
                        }
                        if(tempor_att<(temporizador4-2) && alante==0){
                              alante=1;                          //si se ha
estado 2 segundos concentrado se pone a 1 la bandera que da la orden de mover
el coche
                              alante_tim=temporizador4;
                        }

                  }
```

```
                else{
                        led_orange_off();
                        flag_att=0;
                        tempor_att=0;
                }

                break;

            /* [CODE]: MEDITATION eSense */
            case( 0x05 ):

                value_med=payload[i];                       //se almacena en
value_med el valor de la meditación

                if(value_med>80 && empieza==1){             //si es mayor que
80% se inicia una cuenta para mover el coche hacia atras si se está relajado
                        led_blue_on();                              //durante 3
segundos seguidos

                        if(flag_med==0){
                                tempor_med=temporizador4;
                                flag_med=1;
                        }
                        if(atras_tim<(temporizador4-3)){
                                atras=0;
                        }
                        if(tempor_med<(temporizador4-3) && atras==0){
                                atras=1;                            //si se ha
estado 3 segundos relajado se pone a 1 la bandera que da la orden de mover el
coche
                                atras_tim=temporizador4;
                        }
            }
                else{
                        led_blue_off();
                        flag_med=0;
                        tempor_med=0;

                }

                break;
                }
            }

            i += numBytes;    //se incrementa el índice que recorre la trama
byte a byte
    }

   return( 0 );
}
```

A.13 parsePacketPayload.h

```
#ifndef parsePacketPayload_H
#define parsePacketPayload_H

#include "stm32f4xx.h"

uint8_t parsePacketPayload (void);


#endif
```

A.14 rcc.c

```
#include "stm32f4xx.h"
#include "rcc.h"

/////////* ENABLE - DISABLE AHB1 PERIPHERALS*///////////
void RCC_ENABLE_AHB1_PERIPHERAL (uint32_t PER){
RCC->AHB1ENR |=PER;
}
void RCC_DISABLE_AHB1_PERIPHERAL (uint32_t PER){
RCC->AHB1ENR &=~PER;
}

/////////* ENABLE - DISABLE AHB2 PERIPHERALS*///////////
void RCC_ENABLE_AHB2_PERIPHERAL (uint32_t PER){
RCC->AHB2ENR |=PER;
}
void RCC_DISABLE_AHB2_PERIPHERAL (uint32_t PER){
RCC->AHB2ENR &=~PER;
}

/////////* ENABLE - DISABLE APB1 PERIPHERALS*///////////
void RCC_ENABLE_APB1_PERIPHERAL (uint32_t PER){
RCC->APB1ENR |=PER;
}
void RCC_DISABLE_APB1_PERIPHERAL (uint32_t PER){
RCC->APB1ENR &=~PER;
}

/////////* ENABLE - DISABLE APB2 PERIPHERALS*///////////
void RCC_ENABLE_APB2_PERIPHERAL (uint32_t PER){
RCC->APB2ENR |=PER;
}
void RCC_DISABLE_APB2_PERIPHERAL (uint32_t PER){
RCC->APB2ENR &=~PER;
}
```

A.15 rcc.h

```
#ifndef RCC_H
#define RCC_H

#include "stm32f4xx.h"

/* AHB1 PERIPHERALS */
#define     GPIOA_AHB1      1UL<<0
#define     GPIOB_AHB1      1UL<<1
#define     GPIOC_AHB1      1UL<<2
#define     GPIOD_AHB1      1UL<<3
#define     GPIOE_AHB1      1UL<<4
#define     GPIOF_AHB1      1UL<<5
#define     GPIOG_AHB1      1UL<<6
#define     GPIOH_AHB1      1UL<<7
#define     GPIOI_AHB1      1UL<<8
#define     GPIOJ_AHB1      1UL<<9
#define     GPIOK_AHB1      1UL<<10
#define     CRC_AHB1        1UL<<12
#define     DMA1_AHB1       1UL<<21
#define     DMA2_AHB1       1UL<<22
#define     DMA2D_AHB1      1UL<<23

/* APB1 PERIPHERALS */
#define  UART4_APB1 1UL<<19
#define  USART2_APB1 1UL<<17
#define  TIM7_APB1  1UL<<5
#define  TIM6_APB1 1UL<<4
#define     TIM4_APB1 1UL<<2
#define TIM3_APB1 1UL<<1

/* APB2 PERIPHERALS */
#define  USART1_APB2 1UL<<4
#define  SYSCFG_APB2 1UL<<14


void RCC_ENABLE_AHB1_PERIPHERAL(uint32_t);
void RCC_DISABLE_AHB1_PERIPHERAL(uint32_t);
void RCC_ENABLE_AHB2_PERIPHERAL(uint32_t);
void RCC_DISABLE_AHB2_PERIPHERAL(uint32_t);
void RCC_ENABLE_APB1_PERIPHERAL(uint32_t);
void RCC_DISABLE_APB1_PERIPHERAL(uint32_t);
void RCC_ENABLE_APB2_PERIPHERAL(uint32_t);
void RCC_DISABLE_APB2_PERIPHERAL(uint32_t);

#endif
```

A.16 timer.c

```
/*CAUTION: NOT ALL THE TIMERS SUPPORT THE SAME FEATURES.
READ THE REFERENCE MANUAL*/

#include "stm32f4xx.h"
#include "timer.h"


/* ENABLE - DISABLE TIMER*/
void ENABLE_TIMER(TIM_TypeDef *TIMER){
      TIMER->CR1 |= 0x1;
}

void DISABLE_TIMER(TIM_TypeDef *TIMER){
TIMER->CR1 &= ~0x1;
}

/* SET PRESCALER*/
void SET_TIMER_PRESCALER(TIM_TypeDef *TIMER, uint16_t VALUE){
TIMER->PSC =VALUE;
}

/* UPDATE REGISTER */
void UPDATE_TIMER_REGISTER(TIM_TypeDef *TIMER){
TIMER->EGR |=0x01;
}

/* OVERFLOW*/
void SET_TIMER_OV_INTERRUPT_ENABLE(TIM_TypeDef *TIMER){
      TIMER->SR &= ~0x01;
      TIMER->DIER |= 0x1;
}
void SET_TIMER_OV_INTERRUPT_DISABLE(TIM_TypeDef *TIMER){
      TIMER->SR &= ~0x01;
      TIMER->DIER &= ~0x1;
}


/* COMPARE 1 FUNCTIONS*/
void SET_TIMER_CMP_REG1(TIM_TypeDef *TIMER,uint16_t VALUE){
      TIMER->SR &= ~0x02; // Clear previous channel matches
      TIMER->CCR1 = VALUE;
}

void SET_TIMER_CMP_REG1_INTERRUPT_ENABLE(TIM_TypeDef *TIMER){
      TIMER->DIER |= 0x2;
}

void SET_TIMER_CMP_REG1_INTERRUPT_DISABLE(TIM_TypeDef *TIMER){
      TIMER->DIER &= ~0x2;
}
```

A.17 timer.h

```
#ifndef TIMER_H
#define TIMER_H

#include "stm32f4xx.h"


void ENABLE_TIMER(TIM_TypeDef *);
void DISABLE_TIMER(TIM_TypeDef *);
void SET_TIMER_PRESCALER(TIM_TypeDef *,uint16_t);
void UPDATE_TIMER_REGISTER(TIM_TypeDef *);
void SET_TIMER_CMP_REG1(TIM_TypeDef *,uint16_t);
void SET_TIMER_CMP_REG1_INTERRUPT_ENABLE(TIM_TypeDef *);
void SET_TIMER_CMP_REG1_INTERRUPT_DISABLE(TIM_TypeDef *);
void SET_TIMER_OV_INTERRUPT_ENABLE(TIM_TypeDef *);
void SET_TIMER_OV_INTERRUPT_DISABLE(TIM_TypeDef *);

#endif
```

A.18 uart.c

```
#include "stm32f4xx.h"
#include "uart.h"

#define UART_TX_BUF_SIZE   2048 /**< UART Buffer Size: Must be power of 2 and
less than 256*/
#define UART_TX_BUF_MASK   2047 /**< UART Buffer Mask: UART_TX_BUF_SIZE - 1*/

static uint8_t uart1_tx_buf[UART_TX_BUF_SIZE];  /**< UART1 TX Buffer*/
static uint16_t uart1_tx_tail=0;                /**< UART1 next character to
process*/
static uint16_t uart1_tx_head=0;                /**< UART1 where to store
next character to send*/

static uint8_t uart2_tx_buf[UART_TX_BUF_SIZE];  /**< UART2 TX Buffer*/
static uint16_t uart2_tx_tail=0;                /**< UART2 next character to
process*/
static uint16_t uart2_tx_head=0;                /**< UART2 where to store
next character to send*/

/////////* POINTER FUNCTIONS *////////////

void (*UART1_ON_RX_BYTE)(uint8_t);
void (*UART2_ON_RX_BYTE)(uint8_t);

/////////* UART CONFIGURATION *////////////
extern uint32_t SystemCoreClock;

void UART_ENABLE(USART_TypeDef *USART, uint32_t BAUDRATE, uint32_t MODE){
      uint32_t baudReg;
      uint32_t integerdivider = 0x00;
  uint32_t fractionaldivider = 0x00;
```

```
      USART->CR1          =MODE ;
      USART->CR3        =UART_ERROR_INTERRUPT ;


      integerdivider = ((25 * SystemCoreClock) / (4 * BAUDRATE));

  baudReg = (integerdivider / 100) << 4;

  /* Determine the fractional part */
  fractionaldivider = integerdivider - (100 * (baudReg >> 4));

  /* Implement the fractional part in the register */
  baudReg |= ((((fractionaldivider * 16) + 50) / 100)) & ((uint8_t)0x0F);

  /* Write to USART BRR register */
      USART->BRR        =baudReg;

      /*Enable Uart*/
      USART->CR1       |=1<<13;
}

/////////* RX FUNCTION ASSOCIATION *////////////

void UART1_SET_RX_FUNCTION(void (*PFUNCTION)(uint8_t)){
UART1_ON_RX_BYTE = PFUNCTION;
}

void UART2_SET_RX_FUNCTION(void (*PFUNCTION)(uint8_t)){
UART2_ON_RX_BYTE = PFUNCTION;
}


/////////* INTERRUPT UART1 HANDLER *////////////
void USART1_IRQHandler (void)
{
  /*
  if (USART1->SR & 0x08)                                  //OverRun Error
  {
            USART1->ICR |= 0x08;
            ERROR_HANDLER();
  }

  if (USART1->ISR & 0x04)                                  //Noise Error
  {
    USART1->ICR |= 0x04;
            ERROR_HANDLER();
  }

  if (USART1->ISR & 0x02)                                  //Framing Error
  {
    USART1->ICR |= 0x02;
            ERROR_HANDLER();
  }
  */

      if (USART1->SR & 0x20)                                      //RX
  {
            UART1_RECEIVE((uint8_t)(USART1->DR & (uint8_t)0x00FF));
  }
  else if (USART1->SR & 0x80)                                    //TX
  {
```

```
            if (uart1_tx_tail!=uart1_tx_head)
      {
                USART1->DR=uart1_tx_buf[uart1_tx_tail];
        uart1_tx_tail = (uart1_tx_tail + 1) & UART_TX_BUF_MASK;
      }
      else
      {
        USART1->CR1    &= ~(1<<7);                        //TEI Disabled
      }
    }
 }

 /////////* INTERRUPT UART2 HANDLER *////////////
void USART2_IRQHandler (void)
{
 /*
  if (USART2->ISR & 0x08)                               //OverRun Error
  {
            USART2->ICR |= 0x08;
            ERROR_HANDLER();
  }

  if (USART2->ISR & 0x04)                               //Noise Error
  {
    USART2->ICR |= 0x04;
            ERROR_HANDLER();
  }

  if (USART2->ISR & 0x02)                               //Framing Error
  {
    USART2->ICR |= 0x02;
            ERROR_HANDLER();
  }
     */

  if (USART2->SR & 0x20)                                //RX
  {
            UART2_RECEIVE((uint8_t)(USART2->DR & (uint8_t)0x00FF));
  }
  else if (USART2->SR & 0x80)                                   //TX
  {
    if (uart2_tx_tail!=uart2_tx_head)
      {
                USART2->DR=uart2_tx_buf[uart2_tx_tail];
        uart2_tx_tail = (uart2_tx_tail + 1) & UART_TX_BUF_MASK;

      }
      else
      {
        USART2->CR1    &= ~(1<<7);                        //TEI Disabled
      }
    }
 }

/////////* UART1 SEND BYTE *////////////
void UART1_SEND(uint8_t Buf){
  uint16_t bufTmpIndex;

  bufTmpIndex =(uart1_tx_head + 1) & UART_TX_BUF_MASK;

  if(bufTmpIndex==uart1_tx_tail)
```

```
  {
    OVERFLOW_HANDLER();
  }
  else
  {
    uart1_tx_buf[uart1_tx_head]=Buf;
    uart1_tx_head = bufTmpIndex;

    if ((USART1->CR1  & 0x80)==0)
    {
      USART1->CR1   |= 1<<7;
    }

  }
}

/////////* UART2 SEND BYTE *///////////
void UART2_SEND(uint8_t Buf){
  uint16_t  bufTmpIndex;

  bufTmpIndex =(uart2_tx_head + 1) & UART_TX_BUF_MASK;

  if(bufTmpIndex==uart2_tx_tail)
  {
    OVERFLOW_HANDLER();
  }
  else
  {
    uart2_tx_buf[uart2_tx_head]=Buf;
    uart2_tx_head = bufTmpIndex;

    if ((USART2->CR1  & 0x80)==0)
    {
      USART2->CR1   |= 1<<7;
    }

  }
}

/////////* UART IN TX PROCESS *///////////
uint8_t UART_IN_TX(USART_TypeDef *USART){
      uint8_t status;
      if (USART->CR1 & 1<<7) status=1;
      else status=0;
  return status;
}


/////////* UART1 RECEIVE BYTE *///////////
void UART1_RECEIVE(uint8_t Buf){
      (*UART1_ON_RX_BYTE)(Buf);
}

/////////* UART2 RECEIVE BYTE *///////////
void UART2_RECEIVE(uint8_t Buf){
      (*UART2_ON_RX_BYTE)(Buf);
}

/////////* ERROR MANAGEMENT *///////////

void OVERFLOW_HANDLER(void){
  //NVIC_SystemReset();
```

```
}

void ERROR_HANDLER(void){
/* Write yor code application */
//    led_on(3);
//  NVIC_SystemReset();
}
```

A.19 uart.h

```
#ifndef UART_H
#define UART_H

#include "stm32f4xx.h"

/*Default UART configuration: Tx Enabled, Rx Enable, Ov=16, Tx interrupt
disabled (enabled after), RX interrupt enabled, 1 bit Start, 8 bits data */
#define UART_TX_RX            0x0000002C
#define UART_TX_ONLY          0x00000008
#define UART_ERROR_INTERRUPT  0x00000001

void UART_ENABLE(USART_TypeDef *, uint32_t, uint32_t);
uint8_t UART_IN_TX(USART_TypeDef *);
void UART_DISABLE_RX(USART_TypeDef *);
void UART1_SET_RX_FUNCTION(void (*)(uint8_t));
void UART2_SET_RX_FUNCTION(void (*)(uint8_t));
void UART1_SEND(uint8_t);
void UART2_SEND(uint8_t);
void UART1_RECEIVE(uint8_t);
void UART2_RECEIVE(uint8_t);
void ERROR_HANDLER(void);
void OVERFLOW_HANDLER(void);

#endif
```

A.20 gpio.c

```
#include "stm32f4xx.h"

/////////* PORT/PIN CONFIGURATION *////////////

/* INPUT - OUTPUT - AF - ANALOG*/
void SET_GPIO_AS_INPUT(GPIO_TypeDef *PORT, uint8_t PIN) {
            PORT->MODER  &= ~(1UL<<((PIN<<1)+1));
        PORT->MODER  &= ~(1UL<<(PIN<<1));
}

void SET_GPIO_AS_OUTPUT(GPIO_TypeDef *PORT, uint8_t PIN) {
            PORT->MODER  &= ~(1UL<<((PIN<<1)+1));
        PORT->MODER  |= 1UL<<(PIN<<1);
}

void SET_GPIO_AS_AF(GPIO_TypeDef *PORT, uint8_t PIN, uint8_t AF) {
            PORT->MODER  |= 1UL<<((PIN<<1)+1);
        PORT->MODER  &= ~(1UL<<(PIN<<1));

            if (PIN<=7)
            {
                  PORT->AFR[0]  &= ~(0xF<<(PIN<<2));
                  PORT->AFR[0]  |= AF<<(PIN<<2);
            }
            else
            {
                  PIN=PIN-8;
                  PORT->AFR[1]  &= ~(0xF<<(PIN<<2));
                  PORT->AFR[1]  |= AF<<(PIN<<2);
            }
}

void SET_GPIO_AS_ANALOG(GPIO_TypeDef *PORT, uint8_t PIN) {
            PORT->MODER  |= 1UL<<((PIN<<1)+1);
        PORT->MODER  |= 1UL<<(PIN<<1);
}


/* PULL UP - PULL DOWN*/
void SET_GPIO_PULL_UP(GPIO_TypeDef *PORT, uint8_t PIN) {
            PORT->PUPDR  &= ~(1UL<<((PIN<<1)+1));
        PORT->PUPDR  |= 1UL<<(PIN<<1);
}

void SET_GPIO_PULL_DOWN(GPIO_TypeDef *PORT, uint8_t PIN) {
            PORT->PUPDR  |= 1UL<<((PIN<<1)+1);
        PORT->PUPDR  &= ~(1UL<<(PIN<<1));
}

void SET_GPIO_NO_PULL(GPIO_TypeDef *PORT, uint8_t PIN) {
            PORT->PUPDR  &= ~(1UL<<((PIN<<1)+1));
        PORT->PUPDR  &= ~(1UL<<(PIN<<1));
}

void SET_GPIO_OPEN_DRAIN(GPIO_TypeDef *PORT, uint8_t PIN) {
        PORT->OTYPER  |= 1UL<<(PIN);
```

```
}

/* SPEED */
void SET_GPIO_SPEED_LOW(GPIO_TypeDef *PORT, uint8_t PIN) {
       PORT->OSPEEDR  &= ~(1UL<<(PIN<<1));
}

void SET_GPIO_SPEED_MEDIUM(GPIO_TypeDef *PORT, uint8_t PIN) {
            PORT->OSPEEDR  &= ~(1UL<<((PIN<<1)+1));
        PORT->OSPEEDR  |= 1UL<<(PIN<<1);
}

void SET_GPIO_SPEED_HIGH(GPIO_TypeDef *PORT, uint8_t PIN) {
            PORT->OSPEEDR  |= 1UL<<((PIN<<1)+1);
        PORT->OSPEEDR  |= 1UL<<(PIN<<1);
}


/////////* PORT/PIN ASSIGNMENT *////////////

void GPIO_CLEAR(GPIO_TypeDef *PORT, uint8_t PIN) {
            PORT->BSRR  |= (1UL<<PIN)<<16;
}

void GPIO_SET(GPIO_TypeDef *PORT, uint8_t PIN) {
            PORT->BSRR  |= 1UL<<PIN;
}

void GPIO_TOGGLE(GPIO_TypeDef *PORT, uint8_t PIN) {
            PORT->ODR  ^= 1UL<<PIN;
}

uint8_t GPIO_GET_INPUT(GPIO_TypeDef *PORT, uint8_t PIN) {
      uint8_t value;
      value=(uint8_t)((PORT->IDR)>>PIN)&0x1;
      return value;

}
```

A.21 gpio.h

```
#ifndef GPIO_H
#define GPIO_H

#include "stm32f4xx.h"

/* PIN */
#define PIN0                        0
#define PIN1                        1
#define PIN2                        2
#define PIN3                        3
#define PIN4                        4
#define PIN5                        5
#define PIN6                        6
#define PIN7                        7
#define PIN8                        8
#define PIN9                        9
#define PIN10                       10
#define PIN11                       11
#define PIN12                       12
#define PIN13                       13
#define PIN14                       14
#define PIN15                       15

/* ALTERNATE FUNCTION */
#define AF0                     0
#define AF1                     1
#define AF2                     2
#define AF3                     3
#define AF4                     4
#define AF5                     5
#define AF6                     6
#define AF7                     7

void SET_GPIO_AS_OUTPUT(GPIO_TypeDef *, uint8_t);
void SET_GPIO_AS_INPUT(GPIO_TypeDef *, uint8_t);
void SET_GPIO_AS_AF(GPIO_TypeDef *, uint8_t, uint8_t);
void SET_GPIO_AS_ANALOG(GPIO_TypeDef *, uint8_t);
void SET_GPIO_PULL_UP(GPIO_TypeDef *, uint8_t);
void SET_GPIO_PULL_DOWN(GPIO_TypeDef *, uint8_t);
void SET_GPIO_NO_PULL(GPIO_TypeDef *, uint8_t);
void SET_GPIO_OPEN_DRAIN(GPIO_TypeDef *, uint8_t);
void SET_GPIO_SPEED_LOW(GPIO_TypeDef *, uint8_t);
void SET_GPIO_SPEED_MEDIUM(GPIO_TypeDef *, uint8_t);
void SET_GPIO_SPEED_HIGH(GPIO_TypeDef *, uint8_t);
void GPIO_CLEAR(GPIO_TypeDef *, uint8_t);
void GPIO_SET(GPIO_TypeDef *, uint8_t);
void GPIO_TOGGLE(GPIO_TypeDef *, uint8_t);
uint8_t GPIO_GET_INPUT(GPIO_TypeDef *, uint8_t);

#endif
```

A.22 hexutil.c

```
#include "stm32f4xx.h"

/** HEX_ASCII Table: Table for HEX ASCII conversion */
static uint8_t HEX_ASCII[]="0123456789ABCDEF";

/**
 * Converts one byte to two ascii hexadecimal characters.
 *
 * \param[in]  Byte The byte.
 * \param[out] Buf  The two ascii characters. The buffer must be already
 *                  reserved with at least 2 chars.
 * \return  Void.
 */
void byte_to_hexasciichars(uint8_t Byte, uint8_t *Buf)
{
  uint8_t nibble;

  nibble = (Byte>>4)&0x0F;
  Buf[0] = HEX_ASCII[nibble];

  nibble=Byte&0x0F;
  Buf[1] = HEX_ASCII[nibble];

}


uint8_t hex2bcd (uint8_t x)
{
    uint8_t y;
    y = (x / 10) << 4;
    y = y | (x % 10);
    return (y);
}

uint8_t bcd2hex (uint8_t x)
{
    uint8_t y;
    y = ((x>>4)*10)+(x&0x0F);
    return (y);
}
```

A.23 hexutil.h

```
#ifndef HEXUTIL_H
#define HEXUTIL_H

#include "stm32f4xx.h"

void byte_to_hexasciichars(uint8_t, uint8_t *);
uint8_t hex2bcd (uint8_t x);
uint8_t bcd2hex (uint8_t x);

#endif
```

A.24 print.c

```
#include "stm32f4xx.h"
#include "hexutil.h"
#include "uart.h"
#include "print.h"

/**
 * Print Command.
 * This function send data to the UARTs 2 of the STM32
 *
 * \param[in] Buf  Buffer to send.
 * \param[in] Size Size of the buffer
 * \return Error code ::ERROR_CODE
 */
void print (uint8_t *Buf, uint16_t Size, uint8_t HexFlag){

  int16_t i=0;
  uint8_t auxBuf[2];

      if(!HexFlag)
    {
      for (i=0;i<Size;i++)
      {
        UART2_SEND(Buf[i]);
      }
    }
    else
    {
      for (i=0;i<Size;i++)
      {
        byte_to_hexasciichars(Buf[i], auxBuf);
        UART2_SEND(auxBuf[0]);
        UART2_SEND(auxBuf[1]);
      }

      }
    }



/**
 * Print Command 16 bits.
 * This function send data to the UARTs 2 of the STM32
 *
 * \param[in] Buf  Buffer to send.
 * \param[in] Size Size of the buffer
 * \return Error code ::ERROR_CODE
 */
void print16 (uint16_t *Buf, uint16_t Size){

  int16_t i=0;
  uint8_t auxBuf[4];

  for (i=0;i<Size;i++)
                        {
        byte_to_hexasciichars((uint8_t)(Buf[i]>>8), auxBuf);
        byte_to_hexasciichars((uint8_t)(Buf[i]&0x00FF), auxBuf+2);
        UART2_SEND(auxBuf[0]);
```

```
        UART2_SEND(auxBuf[1]);
                        UART2_SEND(auxBuf[2]);
        UART2_SEND(auxBuf[3]);
      }

}

/**
 * Print Command 32 bits.
 * This function send data to the UARTs 2 of the STM32
 *
 * \param[in] Buf  Buffer to send.
 * \param[in] Size Size of the buffer
 * \return Error code ::ERROR_CODE
 */
void print32 (uint32_t *Buf, uint16_t Size){

  int16_t i=0;
  uint8_t auxBuf[8];

  for (i=0;i<Size;i++)
                        {
                    byte_to_hexasciichars((uint8_t)(Buf[i]>>24), auxBuf);
        byte_to_hexasciichars((uint8_t)((Buf[i]>>16)&0x00FF), auxBuf+2);
        byte_to_hexasciichars((uint8_t)((Buf[i]>>8)&0x00FF), auxBuf+4);
        byte_to_hexasciichars((uint8_t)(Buf[i]&0x00FF), auxBuf+6);
        UART2_SEND(auxBuf[0]);
        UART2_SEND(auxBuf[1]);
                        UART2_SEND(auxBuf[2]);
        UART2_SEND(auxBuf[3]);
                        UART2_SEND(auxBuf[4]);
        UART2_SEND(auxBuf[5]);
                        UART2_SEND(auxBuf[6]);
        UART2_SEND(auxBuf[7]);
      }
}


/**
 * Print Command for digits in BCD digit with decimal part. Up to 5 character
 * This function send data to the UARTs  of the STM32
 *
 * \param[in] Buf  Buffer to send.
 * \param[in] Size Size of the buffer
 * \return Error code ::ERROR_CODE
 */
void print_BCD (int16_t Buf, uint8_t decimalPart){

  uint16_t intPart,rem;
      int8_t  Ndigits;
      int8_t      k;
  uint8_t digits[5];
  uint32_t pow=1;

      if (Buf<0)
      {
            UART2_SEND(0x2D);
            Buf=-Buf;
      }

      if (Buf!=0)
      {
```

```
            for (k=0;k<decimalPart;k++)
            {
                  pow*=10;
            }

            intPart=(uint16_t)(Buf/pow);

            if (intPart==0) Ndigits=decimalPart;
            else if (intPart<10) Ndigits=1+decimalPart;
            else if (intPart<100) Ndigits=2+decimalPart;
            else if (intPart<1000) Ndigits=3+decimalPart;
            else if (intPart<10000) Ndigits=4+decimalPart;
            else Ndigits=5;

            //Calc digits
            digits[0]=(uint8_t)((Buf/10000)+0x30);
            rem=(Buf%10000);
            digits[1]=(uint8_t)((rem/1000)+0x30);
            rem=(rem%1000);
            digits[2]=(uint8_t)((rem/100)+0x30);
            rem=(rem%100);
            digits[3]=(uint8_t)((rem/10)+0x30);
            digits[4]=(uint8_t)((rem%10)+0x30);

            for (k=(5-Ndigits);k<5;k++)
            {
                  if (k==(5-decimalPart))UART2_SEND(0x2E);
                  UART2_SEND(digits[k]);
            }
      }
      else
      {
            UART2_SEND(0x30);
      }

}
```

A.25 print.h

```
#ifndef PRINT_H
#define PRINT_H

#include "stm32f4xx.h"


void print (uint8_t *, uint16_t, uint8_t);
void print16 (uint16_t *, uint16_t);
void print32 (uint32_t *, uint16_t);
void print_BCD (int16_t , uint8_t);


#endif
```

REFERENCIAS

[1] Wikipedia, Cerebro humano [En línea]. Disponible en https://es.wikipedia.org/wiki/Cerebro_humano

[2] Wikipedia, Lóbulo frontal [En línea]. Disponible en https://es.wikipedia.org/wiki/L%C3%B3bulo_frontal

[3] Wikipedia, Lóbulo parietal [En línea]. Disponible en https://es.wikipedia.org/wiki/L%C3%B3bulo_parietal

[4] Wikipedia, Lóbulo occipital [En línea]. Disponible en https://es.wikipedia.org/wiki/L%C3%B3bulo_occipital

[5] Wikipedia, Lóbulo temporal [En línea]. Disponible en https://es.wikipedia.org/wiki/L%C3%B3bulo_temporal

[6] Wikipedia, Neurona [En línea]. Disponible en https://es.wikipedia.org/wiki/Neurona

[7] Wikipedia, Potencial de acción [En línea]. Disponible en https://es.wikipedia.org/wiki/Potencial_de_acci%C3%B3n

[8] HAL, Communication Technologies Based on Brain Activity [En línea]. Disponible en https://hal.inria.fr/file/index/docid/742223/filename/Communication_Technologies_Based_on_Brain_Activity.pdf

[9] "Tema 5 Electroencefalografia", en Instrumentación Biomédica, Alcalá, Departamento de Ingeniería Electrónica, Universidad de Alcalá [En línea]. Disponible en http://www.bioingenieria.edu.ar/academica/catedras/bioingenieria2/archivos/apuntes/tema%205%20-%20electroencefalografia.pdf

[10] Walter J. Freeman, Rodrigo Quian Quiroga (2013), *Imaging Brain Function With EEG Advanced Temporal and Spatial Analysis of Electroencephalographic Signals.*

[11] Ortega, Miguel. (2014) *"EEG: interpretación para mortales", SapiensMedicus* [En línea]. Disponible en http://sapiensmedicus.org/blog/2014/10/21/eeg-interpretacion-para-mortales/

[12] Wikimedia Commons, *Complete neuron cell diagram* [En línea]. Disponible en https://commons.wikimedia.org/wiki/File:Complete_neuron_cell_diagram_es.svg

[13] Wikipedia, Interfaz cerebrocomputadora [En línea]. Disponible en https://es.wikipedia.org/wiki/Interfaz_cerebro-computadora

[14] Wikipedia, Arquitectura BCI [En línea]. Disponible en http://upload.wikimedia.org/wikipedia/commons/3/36/Bci_arquitectura.jpg

[15] Neurosky store [En línea]. Disponible en http://store.neurosky.com/pages/mindwave

[16] Neurosky store [En línea]. Disponible en http://store.neurosky.com/pages/mindwave

[17]Wikipedia, Arquitectura ARM [En línea]. Disponible en https://es.wikipedia.org/wiki/Arquitectura_ARM

[18] Wikipedia, Acorn Computers [En línea]. Disponible en https://en.wikipedia.org/wiki/Acorn_Computers

[19] STMMicroelectronics, STM32F4 Series [En línea]. Disponible en http://www2.st.com/content/st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32f4-series.html?querycriteria=productId=SS1577

[20] STMMicroelectronics, STM32F4Discovery [En línea]. Disponible en http://www2.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-discovery-kits/stm32f4discovery.html

[21] ARM, *CortexM4chipdiagramL*G [En línea]. Disponible en http://www.arm.com/assets/images/processor/Cortex-M4-chip-diagram-LG.png

[22] "CortexM4 Devices. Generic User Guide", en infocenter ARM [En línea]. Disponible en http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf

[23] "CortexM4 Revision r0p0. Technical Reference Manual", en infocenter ARM [En línea]. Disponible en http://infocenter.arm.com/help/topic/com.arm.doc.ddi0439b/DDI0439B_cortex_m4_r0p0_trm.pdf

[24] "STM32F405xx/STM32F407xx Datasheet", en STMMicroelectronics [En línea]. Disponible en http://www2.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

[25] STMMicroelectronics, STM32F4Discovery [En línea]. Disponible en http://www2.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-discovery-kits/stm32f4discovery.html#design-scroll

[26] Wiki, Serial Port Bluetooth Module (Master/Slave): HC05 [En línea]. Disponible en http://wiki.iteadstudio.com/Serial_Port_Bluetooth_Module_(Master/Slave)_:_HC-05

[27] "ThinkGear Serial Stream Guide", en Neurosky [En línea]. Disponible en http://developer.neurosky.com/docs/doku.php?id=thinkgear_communications_protocol

[28] "Tx2S datasheet", en Radiometrix [En línea]. Disponible en http://www.radiometrix.com/files/additional/tx2s.pdf

[29] Wikipedia, Ultra High Frecuency [En línea]. Disponible en https://en.wikipedia.org/wiki/Ultra_high_frequency