

Real-Time Audio Processor

Memoria del proyecto

RUBÉN AGUSTÍN
DAVID ANDRINO
ESTELA MORA
FERNANDO SANZ

Ingeniería de Sistemas Electrónicos
Primavera 2024

Índice

1. Introducción	1
1.1. Especificaciones iniciales del sistema a diseñar y construir	1
1.2. Especificaciones finales del sistema diseñado y construido	1
2. Desarrollo de subsistemas	3
2.1. La placa	3
2.1.1. Algunas consideraciones	3
2.2. Subsistema de alimentación	4
2.2.1. Circuito de carga	4
2.2.2. Convertidor elevador	6
2.2.3. Medidor de consumo	7
2.2.4. Diseño de PCB	8
2.3. Subsistema de audio	10
2.3.1. Divisor de rail	10
2.3.2. Habilitación del circuito	10
2.3.3. Multiplexación de audio	11
2.3.4. Cambiador de nivel lógico	12
2.3.5. Amplificador de audio para los auriculares	12
2.3.6. Amplificador de altavoces	14
2.3.7. Diseño de PCB	14
2.4. Módulo de radio	15
2.5. Módulo MP3	16
2.6. Módulo NFC	16
2.7. Entre Dos Tierras	17
2.8. Conexión de la alimentación a la placa	19
3. Software	20
3.1. Interfaz de usuario	20
3.1.1. Interfaz web	20
3.2. Interfaz táctil	23
3.2.1. Inicio	23
3.2.2. Radio	24
3.2.3. MP3	25
3.2.4. Filtros	26
3.3. Módulos software	26
3.3.1. Módulo de LCD	27
3.3.2. Módulo de control	30
3.3.3. Módulo de control de consumo	32
3.3.4. Módulo de acceso concurrente al I2C	32
3.3.5. Módulo de procesado digital	32
3.3.6. Módulo NFC	37
3.3.7. Módulo de la SD	39
3.3.8. Módulo RTC	40
3.3.9. Módulo Radio	40
3.3.10. Módulo MP3	41
3.3.11. Módulo Web	41
4. Depuración y test	44
4.1. Pruebas software	44
4.1.1. Pruebas del módulo de procesado digital de señal	44
4.1.2. Test Radio	44
4.1.3. Test MP3	45
4.1.4. Test RTC	45
4.1.5. Test Web	45
4.2. Pruebas hardware	47

4.2.1. Pruebas de la placa de audio	47
4.2.2. Pruebas del circuito de alimentación	48
5. Presupuesto final	50
6. Equipo de trabajo	51
7. Acrónimos utilizados	52
8. Bibliografía	53
Anexo A. Webench Design Report	54
Anexo B. Circuito de alimentación	61
Anexo C. Circuito de alimentación	64
Anexo D. Estructura de los mensajes de control	65
Anexo E. Ficheros de adaptación de la pantalla	67

Índice de figuras

1. Sistema Completo	2
2. Diagrama de bloques hardware del sistema	3
3. Ciclo de carga de una batería de ión de litio	4
4. Diagrama de bloques interno del BQ25606	5
5. Subcircuito de carga de batería	6
6. Diagrama de bloques interno del LM51561H	7
7. Circuito del convertidor elevador	8
8. Circuito medidor de consumo	8
9. Circuito de alimentación	9
10. Circuito divisor de raíl	10
11. Circuito de habilitación	11
12. Puerta de transmisión con transistores con canal desconectado	11
13. Circuito cambiador de niveles	12
14. Foto del circuito cambiador de nivel	13
15. Circuito amplificador de auriculares	13
16. Circuito amplificador de altavoces	14
17. Circuito de audio completo	15
18. Sintonizador FM RDA5807M	16
19. Reproductor MP3 YX5300	16
20. Módulo NFC ANT7-T-M24SR64	17
21. Esquemático del circuito sumador	18
22. Foto del circuito sumador analógico	18
23. Página Principal	20
24. Página Radio	21
25. Página MP3	22
26. Página Filtros	22
27. Pantalla principal de la aplicación	24
28. Pantalla de la radio	25
29. Pantalla del MP3	26
30. Pantalla de los filtros	26
31. Diagrama de bloques software del proyecto	27
32. Mecanismo de doble <i>buffer</i> para audio	34
33. Diagrama de flujo de una operación de lectura con el NFC	38
34. Configuración del sistema implementando FatFs	39
35. Camino de audio sin procesado	44

36.	Señal cuadrada procesada digitalmente	44
37.	Respuesta del circuito amplificador de auriculares	47
38.	Respuesta del circuito amplificador de altavoces	47
39.	Regulación de carga del circuito de baterías	48
40.	Ruido en la señal de alimentación	49

Índice de fragmentos

1.	Estructura para los mensajes al control	30
2.	Configuración del TRGO del <i>timer</i> de sincronización	33
3.	Función de procesado de audio	37
4.	Ejemplo archivo .CGI	41
5.	Ejemplo procesamiento archivo .CGI	42
6.	Función updatePeriodic()	42
7.	Archivo time.cgi	43
8.	Fichero controlThread.h con las estructuras de mensajes	65
9.	Fichero tft.h	67
10.	Fichero tft.c	68
11.	Fichero Touchpad.h con las estructuras de mensajes	77
12.	Fichero Touchpad.c	78

1. Introducción

1.1. Especificaciones iniciales del sistema a diseñar y construir

El proyecto consiste en el desarrollo de un sistema de procesamiento de audio en tiempo real con un sistema centrado en un microprocesador.

El sistema permitirá entrada de audio a través de un módulo de radio FM y la lectura de canciones de una tarjeta microSD externa. Tras capturar dicho audio, se le aplicará un procesamiento digital implementado mediante la librería CMSIS DSP y se reproducirá por un altavoz o unos auriculares.

La interacción con el sistema se realizará a través de una pantalla táctil o una interfaz web, la cual requiere conexión a través de Ethernet. Dicha interfaz permite seleccionar la entrada de audio, controlar el volumen y ecualización de audio y elegir la salida. Además, permite controlar la emisora de radio sintonizada.

El sistema almacenará los parámetros seleccionados (entrada, salida y filtros) y una lista de emisoras favoritas en la tarjeta microSD, los cuales se cargarán al iniciar el equipo.

El sistema será completamente autónomo, contando con una batería con su correspondiente circuitería de carga, protección y medición de consumo. Se ofrecerá información sobre la batería en la interfaz gráfica del sistema. Además, el sistema contará con un modo de bajo consumo para alargar la duración de dicha batería.

Para la selección de canciones se permitirá el uso de tarjetas NFC preconfiguradas con canciones o emisoras preconfiguradas, para la interacción sin interfaz gráfica. Por último, el sistema utilizará el RTC integrado en la placa para mantener la hora y el protocolo SNTP para la sincronización.

1.2. Especificaciones finales del sistema diseñado y construido

El sistema permite la entrada de audio a través del Sintonizador FM y la lectura de canciones de una tarjeta microSD externa mediante un reproductor MP3. Tras capturar dicho audio, se le aplica el procesamiento digital implementado mediante la librería CMSIS DSP y se reproduce por un altavoz o unos auriculares.

La interacción con el sistema se realiza a través de la pantalla táctil o mediante la interfaz web, la cual requiere conexión a través de Ethernet. Dicha interfaz nos permite seleccionar la entrada de audio, controlar el volumen y aplicar la ecualización deseada al audio y elegir la salida. Además, permite seleccionar y controlar la emisora de radio sintonizada.

El sistema permite la lectura de los parámetros de configuración seleccionados (entrada, salida, filtros y volumen) y una lista de canciones de la tarjeta uSD, escritos mediante un ordenador.

Por otra parte, el sistema permite almacenar un mapa de frecuencia a emisora en el almacenamiento de la propia tarjeta.

El sistema no es completamente autónomo, ya que, aunque cuente con una batería con su correspondiente circuitería de carga, protección y medición de consumo, el sistema necesita una conexión constante mediante el ST_Link. Se ofrece información sobre el consumo medido en la interfaz gráfica del sistema. Además, el sistema cuenta con un modo de bajo consumo para alargar la duración de dicha batería.

Para la selección de canciones y emisoras, se permitirá el uso de tarjetas NFC preconfiguradas con canciones o emisoras preconfiguradas, para la interacción sin interfaz gráfica. Por último, el sistema utiliza el RTC integrado en la propia placa para mantener la hora y fecha actual mediante la sincronización con un servidor SNTP.

A continuación se adjunta una imagen del sistema completo, con todos sus componentes y en su total funcionamiento:

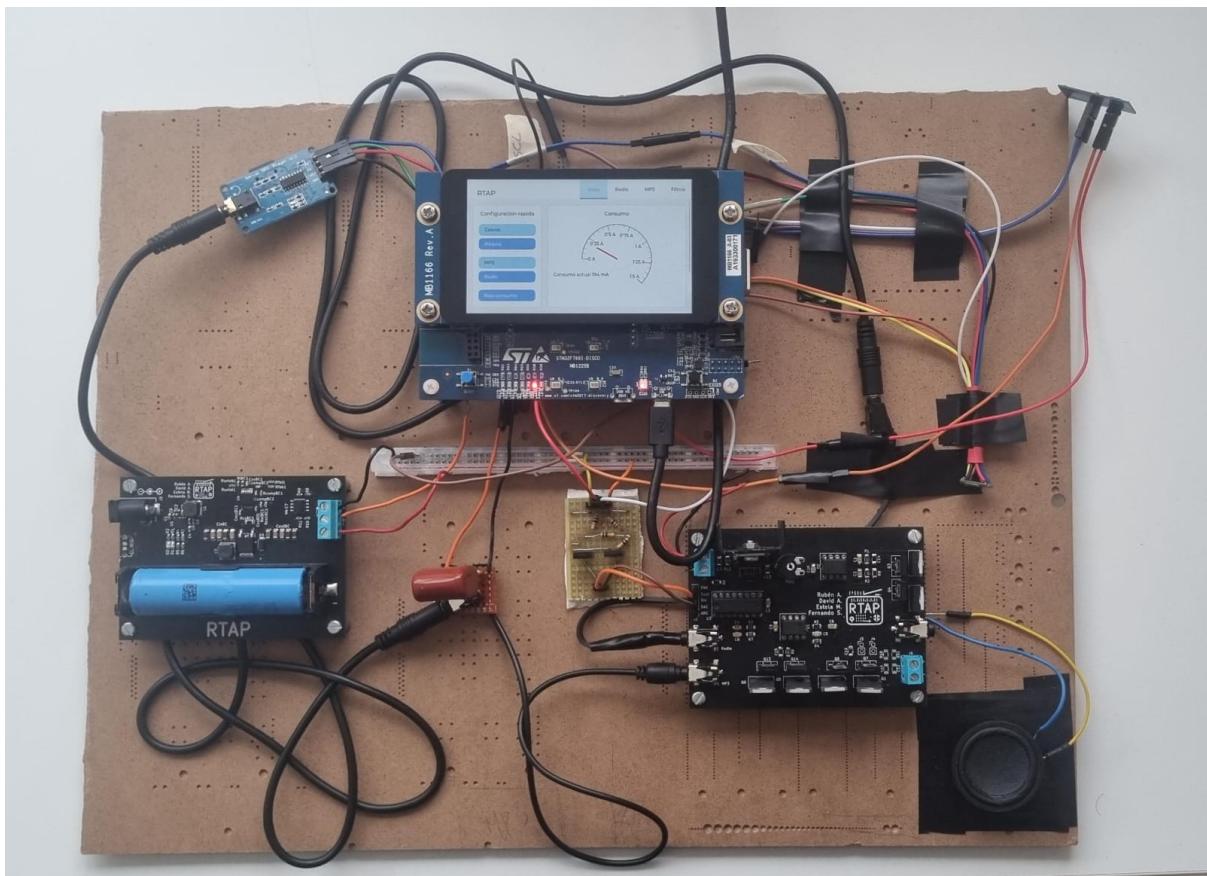


Figura 1: Sistema Completo

2. Desarrollo de subsistemas

Para este proyecto hemos desarrollado dos subsistemas analógicos propios, sobre dos PCB distintas, una de alimentación y un amplificador de audio. Además, hemos utilizado tres subsistemas ya existentes para la recepción de audio, la reproducción de música en MP3 y la lectura de información NFC de un dispositivo móvil.

Se puede ver un diagrama de bloques hardware en la siguiente figura:

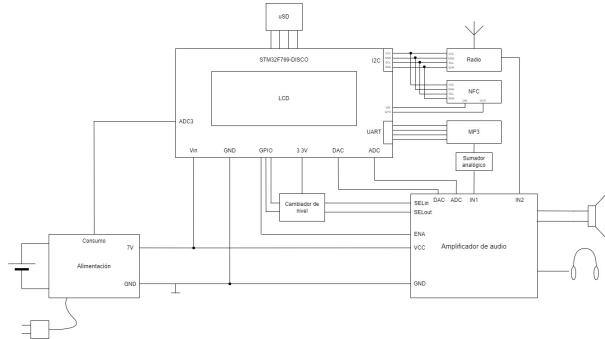


Figura 2: Diagrama de bloques hardware del sistema

2.1. La placa

Para este proyecto se ha utilizado la placa STM32F769NI-DISCO, que es una placa de desarrollo comercializada por STM. Dispone de un procesador ARM Cortex M7, que puede funcionar a una frecuencia de hasta 216 MHz.

Dispone de pines dispuestos de una forma adecuada para ser compatible con Arduino Uno.

Entre los periféricos presentes en la placa, los más importantes que utilizamos son:

- **La pantalla:** Dispone de una pantalla LCD táctil de 4.3 pulgadas, accesible a través del periférico DSI, con una resolución de 800 x 480 píxeles, con una profundidad del color de 16 bits, lo que resulta en un total de 2^{16} colores posibles de representar. En concreto, se asignan 5 bits para las componentes roja y azul, siendo los 6 restantes para la verde, que es la más importante.
- **La DMA2:** La pantalla restringe el uso de las DMA, siendo la 2 la única opción posible, puesto que es la única que tiene la opción de *Memory-To-Memory*. Cualquier canal y flujo (*Channel* y *Stream*) son de libre elección.
- **Los ADCs:** La placa dispone tres ADCs, que funcionan a una frecuencia de 48 kHz, de los cuales nuestro proyecto utiliza dos: el ADC1 se utiliza para medir el consumo, y el ADC3 se utiliza como pin de entrada para el audio.
- **El DAC:** La placa dispone de un DAC de 2 canales. En la documentación de la placa no se anuncia esta capacidad, únicamente se puede encontrar en la documentación del microcontrolador. El DAC también funciona a 48kHz, y se sincroniza con el ADC, como se explica en Párrafo 3.3.5.1.
- **El I2C:** Utilizamos un I2C para la comunicación con el módulo del NFC.
- **La SD:** Dispone de un puerto para introducir una tarjeta SD, a través del periférico SDMMC.

2.1.1. Algunas consideraciones

En algunos proyectos, hemos intentado utilizar la sentencia `printf`, pero a priori parecía una tarea imposible. Revisando la documentación, constatamos que el pin SW0 está desconectado en la configuración por defecto. Para que funcione dicha función, hay que soldar R92, que conecta SW0 con el pin adecuado. Esta resistencia funciona como un jumper, ya que el valor anunciado por el fabricante es de 0 Ω.

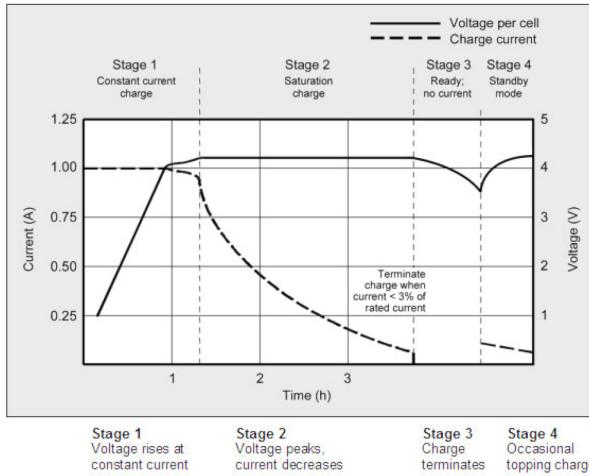
Tras sujetar un cable de forma manual, conseguimos visualizar la salida de la UART desde el panel del debugger.

2.2. Subsistema de alimentación

El subsistema de alimentación es el encargado de permitir que el sistema se alimente a través de baterías. También permite cargar la batería e incluso ambas acciones simultáneamente.

Como batería hemos decidido utilizar una batería de ion de litio 18650, un modelo bastante estándar en las aplicaciones integradas. Concretamente, hemos decidido utilizar el modelo INR18650-29E de Samsung, que tiene una tensión nominal de 3.7 V y una capacidad de 2900 mAh.

Sin embargo, este tipo de baterías tienen unos requisitos de carga muy estrictos. Contando con que la batería parte de un estado correcto (no por debajo del límite de tensión segura), se debe cargar primero con un método de corriente constante hasta alcanzar una tensión cercana a la máxima y después cambiar a un método de tensión constante hasta finalizar la carga [1]. Un ejemplo de ciclo de carga se puede ver en la Figura 3.



Si no se respeta el ciclo de carga de estas baterías, se tiene una alta probabilidad de que falle de forma violenta, siendo incluso un peligro de incendio.

2.2.1. Circuito de carga

Para facilitarnos la tarea de respetar este ciclo hemos utilizado una solución integrada de Texas Instruments, el BQ25606, un cargador de una celda de litio que soporta hasta 3 amperios de corriente. [2]

Gracias a este circuito integrado y bastantes componentes externos, podemos diseñar un circuito que, a partir de una tensión de entrada entre 5 y 12 voltios y utilizando un convertidor reductor, carga la batería adecuadamente y de forma segura.

La entrada al circuito puede ser a través de un conector *Jack* de alimentación o un conector *Micro USB*. Además, si se utiliza el segundo conector, el integrado se encarga de negociar el protocolo de carga rápida con la fuente de alimentación para incrementar la corriente de entrada y mejorar la potencia de carga.

Además, si el circuito está conectado a la alimentación y la fuente tiene suficiente capacidad, la corriente de salida se obtiene de la entrada en lugar de la batería, gracias a la tecnología PowerPath de TI. Esta tecnología además permite balancear las tres corrientes, por lo que si el sistema requiriera de más corriente de la que la fuente de alimentación pudiera proveer, se obtendría también de la batería realizando un

esfuerzo coordinado entre la fuente y la batería. Si por el contrario la fuente puede ofrecer más corriente de la que se está solicitando en la salida, se utiliza este excedente para cargar la batería.

Este circuito cuenta además con dos indicadores LED que informan sobre si la fuente de alimentación está conectada y las tensiones son correctas (verde en nuestro circuito) y si se está cargando la batería o hay algún fallo (roja en nuestro circuito). La segunda luz se mantiene encendida mientras se está cargando, se apaga cuando se ha finalizado la carga y parpadea a 1 Hz de frecuencia si hay algún error.

Para ofrecer todas estas características, el circuito integrado consta de una máquina de estados finitos y múltiples comparadores de error. Con esta inteligencia conmuta tres transistores para conectar la batería y para gestionar el convertidor conmutado síncrono que se construye. Se puede ver un diagrama de bloques resumido del circuito (obtenido de la hoja de catálogo) en la Figura 4.

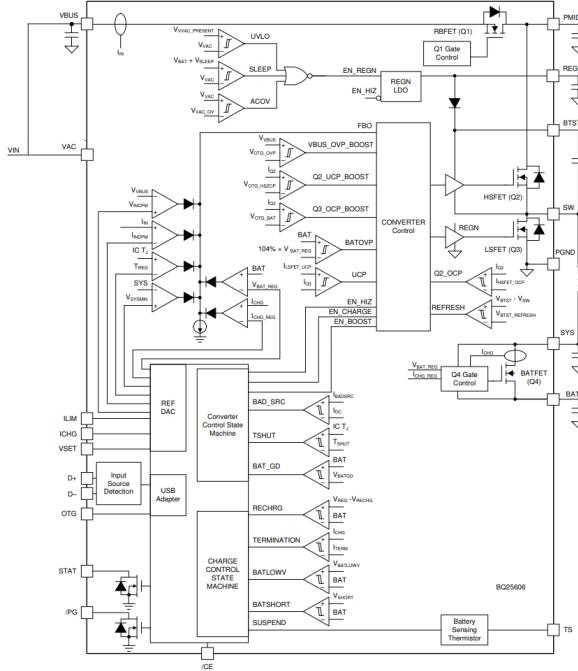


Figura 4: Diagrama de bloques interno del BQ25606

Este circuito ofrece a su salida una tensión aproximadamente igual a la de la batería durante el funcionamiento normal, por lo que está alrededor de los 3.7 V. Sin embargo, si la tensión de la batería cae por debajo de 3.5 V, el circuito se encarga de ofrecer dicha tensión a la salida, por lo que nunca bajará de dicho valor (mientras la batería puede ofrecer corriente y no esté descargada).

El circuito integrado ofrece también la posibilidad de utilizar una batería con sensor de temperatura integrado, pero no vamos a utilizarlo debido al incremento en coste de la misma.

En la hoja de catálogo del integrado se ofrece información sobre el proceso de diseño de un circuito alrededor de dicho integrado. Además, se tiene una nota de aplicación sobre el diseño de circuitos alrededor de este integrado [3]:

- Inductancia de $2.2 \mu F$ para reducir el rizado de corriente.
- Pin VSET flotante para tensión de carga máxima de 4.208 V.
- Divisor de tensión de dos resistencias de $10 k\Omega$ en TS para no utilizar divisor de tensión.
- Resistencia de 165Ω en ILIM para limitar la corriente de entrada cuando no se negocia carga rápida a 3 A (normalmente se utiliza mucho menos).
- Resistencia de 487Ω en ICHG para limitar la corriente de carga máxima a 1.4 A como se indica en las especificaciones de la batería.
- El resto de componentes se toman de la Figura 10-1 de la hoja de catálogo.

Por tanto, esta parte del circuito queda como se puede ver en la Figura 5.

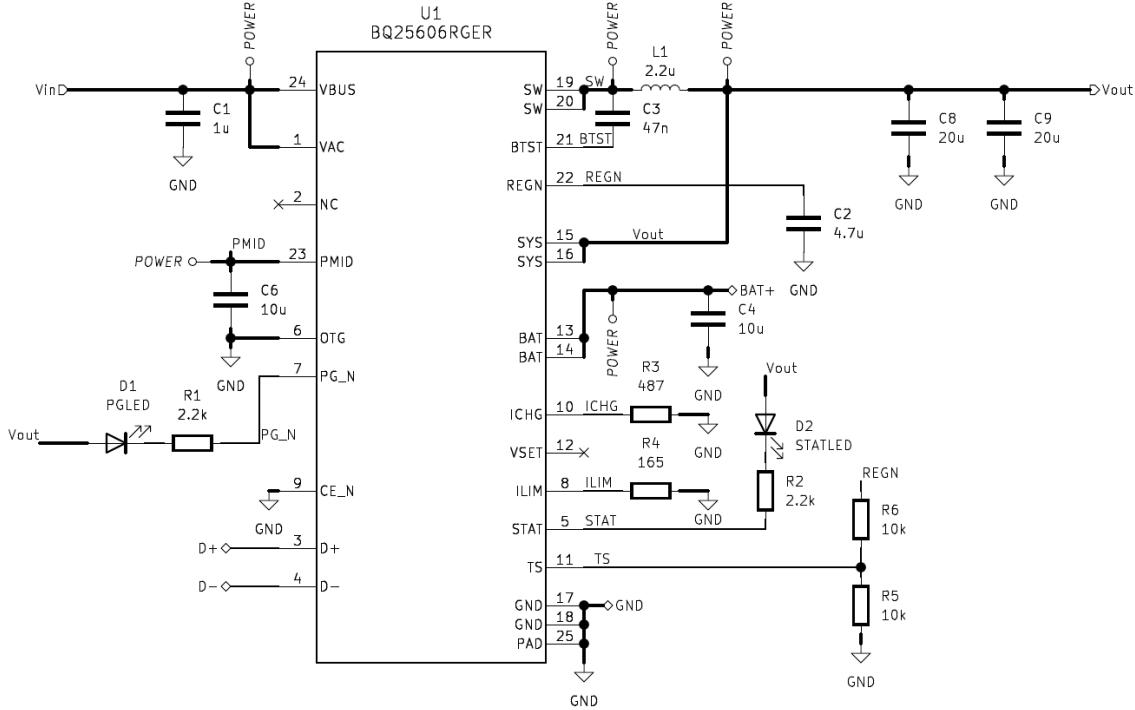


Figura 5: Subcircuito de carga de batería

2.2.2. Convertidor elevador

La placa que utilizamos especifica una tensión de entrada de 7 V a 12 V si se utiliza el pin V_{in} inferior. Preferimos utilizar este método ya que la entrada de 5 V no tiene protección y se puede dañar la placa si no se realiza todo el proceso adecuadamente.

Experimentalmente hemos notado que la placa utiliza la misma corriente de entrada para cualquier valor de tensión, por lo que seguramente utilice un convertidor de tensión lineal internamente para generar las tensiones. Por tanto, hemos preferido tomar el valor más pequeño de las tensiones de entrada para reducir la pérdida de potencia.

La tensión de salida del circuito de carga oscila entre los 3.5 V y los 4.2 V, por lo que hemos diseñado un convertidor conmutado elevador o *Boost Converter* para obtener dicha tensión a partir de la salida del cargador.

Un circuito convertidor elevador está compuesto básicamente por una bobina, un transistor y un controlador PWM. Como controlador PWM hemos utilizado otra solución integrada de Texas Instruments debido a la alta precisión que nos permite tener, ya que un fallo en este circuito podría dañar seriamente al resto del sistema.

El circuito integrado elegido es el LM51561H. Este circuito es un controlador de convertidores conmutados de tipo *Boost*, *SEPIC* y *Flyback* con gran rango de tensiones y muy elevada protección. [4]

Además, indica específicamente que está pensado para aplicaciones de batería gracias a su baja tensión de entrada necesaria para funcionar. Al contrario que el circuito del cargador, este integrado no incluye los transistores internos por lo que se tienen que añadir externamente.

Este integrado consta básicamente de comparadores de error respecto a tensiones de referencia y un generador de señales triangulares para la generación de PWM. Cuenta además con la posibilidad de compensar el circuito para evitar su inestabilidad.

La versión LM51561H es igual que la LM5156H solo que con protección en modo *Hiccup*. En este modo de protección, si se detecta una sobrecarga, se desactiva el convertidor y se comienza una espera. Al finalizar la espera, se trata de volver a arrancar la conversión. Si la condición de sobrecarga sigue presente, se

vuelve a dormir y repetir el ciclo. Esto permite una mucho menor potencia media en condición de sobrecarga en comparación a la protección por corriente media, pico o valle. [5]

Se puede ver un diagrama de bloques resumido en la Figura 6.

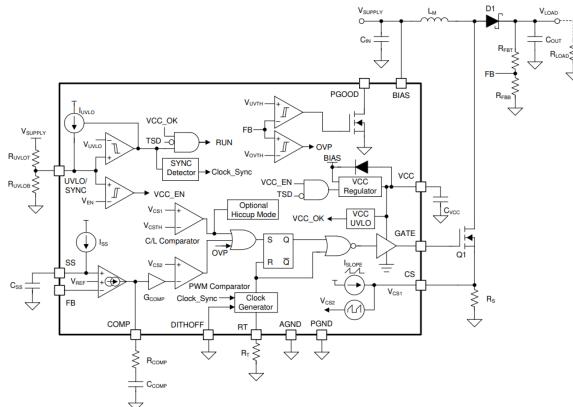


Figura 6: Diagrama de bloques interno del LM51561H

Algunas de las características destacables de nuestro circuito son:

1. Protección contra infravoltaje (*Undervoltage Lockout*). Se comprueba que el pin UVLO tenga una tensión superior a 0.55 V para comenzar la configuración interna y superior a 1.5 V para comenzar a comutar.
2. Frecuencia elevada de conmutación para evitar interferencias en las bandas de AM y reducir las pérdidas de conmutación.
3. Salida de 7 V muy estable y con poca dependencia de la corriente de salida.

Para el diseño de este circuito, el fabricante recomienda el uso de la herramienta WEBENCH Power Designer¹, que permite reducir la complejidad del diseño realizando los cálculos necesarios para los parámetros deseados y la compensación de la función de transferencia del circuito.

Utilizando dicha herramienta, generamos el circuito que hemos utilizado con los siguientes parámetros:

1. $V_{in,\min} = 3,5 \text{ V}$
2. $V_{in,\max} = 4,2 \text{ V}$
3. $V_{out} = 7,0 \text{ V}$
4. $I_{out} = 1,0 \text{ A}$

Una vez introducidos los parámetros, el software genera el circuito con los componentes recomendados, permitiendo sustituirlos por equivalentes y generando las gráficas de funcionamiento aproximado. Se puede encontrar el reporte generado en el Apéndice A.

Con el resultado de dicho software, se construye un circuito como el de la Figura 7.

2.2.3. Medidor de consumo

Otra parte importante de este circuito es el medidor de consumo integrado. Para ello, primero pensamos en construir un amplificador de instrumentación, que se puede adquirir ya implementado en un mismo paquete o realizar con dos amplificadores operacionales. Sin embargo, durante la búsqueda de modelo de amplificador operacional encontramos un modelo que se ofrece como especializado en medición de corriente en el nivel bajo (*Low-side current switching*) por lo que decidimos quedarnos con él.

Dicho integrado es el OPA187ID, un amplificador operacional de precisión con aproximadamente cero tensión de offset ($10 \mu\text{V}$, $0,001 \mu\text{V}/^\circ\text{C}$). [6]

¹<https://www.ti.com/tool/download/SNVC224>

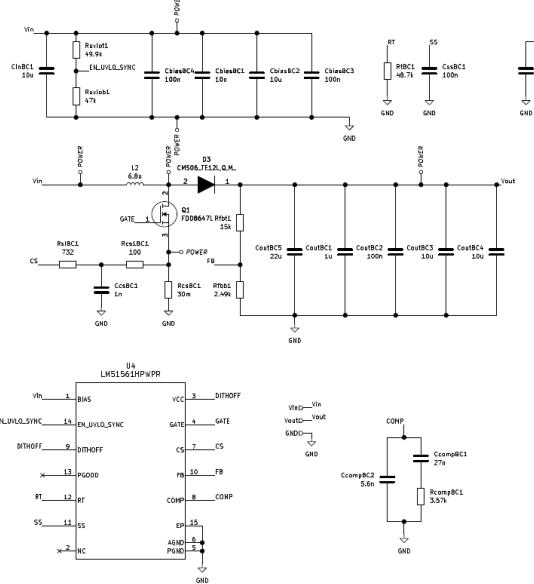


Figura 7: Circuito del convertidor elevador

Dicho amplificador operacional recomienda montar una configuración de amplificador diferencial con una resistencia *shunt* de muy baja impedancia y configurar la ganancia deseada mediante las resistencias de realimentación.

Como resistencia de medida hemos elegido un valor bajo, de $10\text{ m}\Omega$ y que soporta 1 W de potencia. Para simplificar los cálculos, hemos decidido asignar el rango de entrada de un ADC de nuestra placa ($[0, 3,3]\text{ V}$) a un rango de $[0, 3]\text{ A}$. Por ello, la ganancia es:

$$G = \frac{3,3 - 0}{3 - 0} = 1,1\text{V/A}$$

Para conseguir dicha ganancia es tan sencillo como utilizar resistencias con un cociente de 110, por lo que elegimos valores de $1\text{ k}\Omega$ y $110\text{ k}\Omega$ en el circuito, por lo que obtenemos el circuito de la Figura 8.

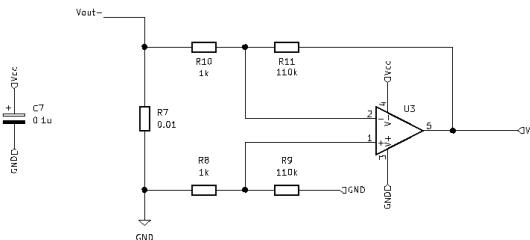


Figura 8: Circuito medidor de consumo

2.2.4. Diseño de PCB

Los tres circuitos se han integrado en una sola PCB, permitiendo una mucho mejor integración y uso mucho más sencillo. Se ha integrado un zócalo para insertar la batería sobre la placa y evitar la posibilidad de una mala conexión. Además, se han incluido los conectores de entrada de potencia previamente mencionados, tanto el Jack de alimentación como el conector micro-USB. Por otro lado, la salida del sistema se realiza a través de unos terminales de tornillo que incluyen la tensión de alimentación positiva, negativa y la medida de consumo.

Sin embargo, por un problema durante el ensamblaje de la placa, se destruyeron los pads del conector micro-USB, por lo que el conector no funciona correctamente. Sin embargo, el sistema funciona perfectamente a

través del Jack de alimentación.

La alta complejidad de los circuitos integrados hace que sus empaquetados sean muy pequeños, por lo que la soldadura se ha tenido que realizar mediante horno y con stencil. Sin embargo, esto nos ha permitido colocar los componentes muy próximos entre sí, reduciendo el ruido electromagnético generado y las componentes parásitas del circuito. Concretamente, el circuito integrado de carga es de empaquetado VQFN24 con un tamaño de $4 \times 4 \text{ mm}$ y el integrado del conversor es HTSSOP14 con un tamaño de $5 \times 4,4 \text{ mm}$.

Además, el pequeño tamaño de estos circuitos reduce significativamente su capacidad de disipación térmica. Por ello, ambos incluyen un pad en su parte inferior que debe ser soldado a un pad que además incluya muchas vias térmicas a la capa inferior, en la que debe haber un plano de tierra grande para disipar el calor que se genere. Por ello y para ofrecer un camino de retorno a las altas corrientes que recorren este circuito, la capa inferior se dedica casi exclusivamente a la masa del circuito.

Ambos circuitos integrados ofrecen indicaciones sobre la disposición de los componentes que se han seguido rigurosamente. Por ejemplo, se recomienda mantener los bucles de commutación lo más pequeños posibles o mantener separadas las tierras digital y analógica, juntandolas únicamente en un punto.

Ciertas zonas del circuito han tenido que ser diseñadas mediante polígonos especiales al no poderse realizar la conexión mediante pistas normales o necesitar zonas especialmente grandes para caminos de mucha corriente.

Se puede ver el resultado final del circuito de alimentación en la Figura 9.

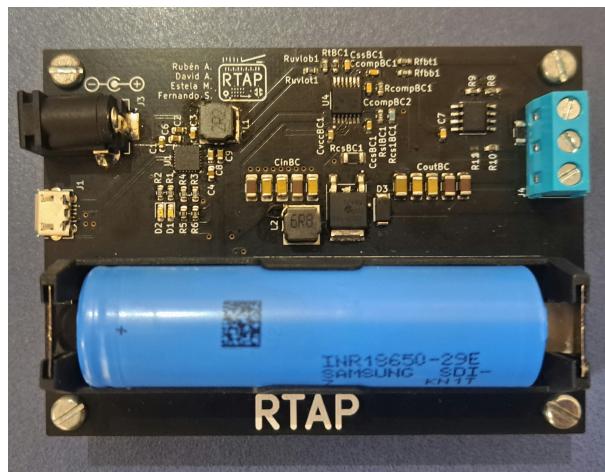


Figura 9: Circuito de alimentación

Se tiene un esquema completo del circuito en el Apéndice B.

2.3. Subsistema de audio

El subsistema analógico de audio tiene tres funciones principales:

1. Multiplexar las dos entradas de audio (MP3 y Radio) a un ADC del microcontrolador.
2. Multiplexar la salida proveniente del DAC a una de las dos salidas de audio (Altavoces o auriculares).
3. Amplificar la salida de audio seleccionada.

2.3.1. Divisor de rail

Este sistema se alimenta directamente desde el módulo de alimentación explicado en el apartado anterior. Por ello, cuenta con una entrada unipolar de 7 voltios. Sin embargo, los circuitos de audio necesitan alimentación bipolar para funcionar. Por ello, hemos decidido implementar un circuito divisor de rail, que genera una tensión intermedia que se utilizará como nueva referencia del circuito, obteniendo una alimentación virtual de ± 3.5 V.

El circuito utilizado es un divisor de tensión de precisión implementado mediante el amplificador operacional OP07. Este amplificador operacional tiene muy bajo error estático y además cuenta con dos terminales para el ajuste del offset mediante un potenciómetro, por lo que es ideal para nuestra aplicación.

Sin embargo, los amplificadores operacionales permiten muy poca corriente a través de su terminal de salida, por lo que se necesita un circuito que permita incrementar la corriente de salida o entrada de dicho circuito sin afectar demasiado a la salida. Para ello, se utiliza una topología *Push-Pull* en la que se utilizan dos pares de Darlington, es decir, cuatro transistores, para incrementar la capacidad de corriente del circuito.

Al introducirlos dentro del lazo de realimentación, se elimina el efecto de las tensiones de base y se elimina el ruido que pudieran introducir, pero a cambio introducen la posibilidad de inestabilizar el circuito. Teniendo esta posibilidad en cuenta, incluimos la posibilidad de soldar un condensador entre el terminal de salida del amplificador y la alimentación negativa del circuito, para introducir un polo que compense la estabilidad. Sin embargo, hemos acabado no necesitando utilizarlo.

Por tanto, el circuito generador de tierra virtual o divisor de rail queda como se puede ver en la Figura 10.

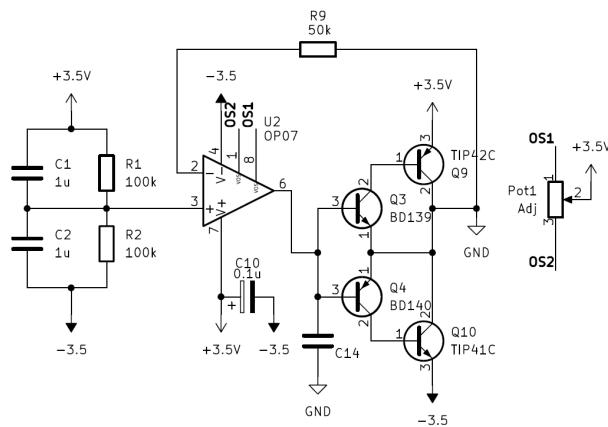


Figura 10: Circuito divisor de rail

2.3.2. Habilitación del circuito

Para eliminar el consumo parásito del circuito cuando el sistema entra en el modo bajo consumo, se ha implementado un subcircuito de habilitación el cual permite encender o apagar el resto de subsistema (aunque finalmente el consumo parásito es muy pequeño).

Dicho circuito consiste en un transistor MOSFET de canal N en la alimentación, que permite cortar o dejar pasar la alimentación. Además, la baja impedancia de conducción del transistor permite que no haya casi pérdidas de potencia en el transistor.

Sin embargo, ya que para cortar el transistor se necesita polarizar la puerta con una tensión próxima a 7 voltios y soportar las corrientes de los transitorios de commutación, se utiliza otro transistor con una resistencia de *pull-up* para adaptar los niveles los GPIO y reducir la corriente necesaria. Esto tiene el efecto añadido de invertir la polaridad de la habilitación que junto a la inversión del canal N se anulan, provocando que un nivel alto en el GPIO habilite el circuito.

Por tanto, el circuito final es el que se ve en la Figura 11.

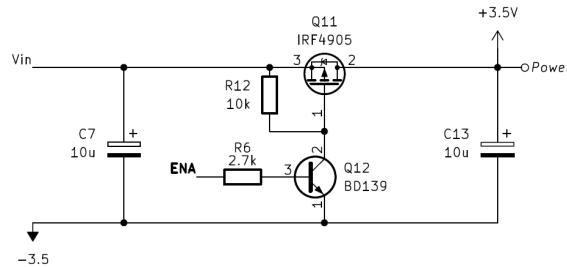


Figura 11: Circuito de habilitación

2.3.3. Multiplexación de audio

Para la multiplexación de audio se va a utilizar un multiplexor integrado. Inicialmente tratamos de diseñar un circuito que multiplexara los caminos de audio mediante componentes discretos, encontrando la estructura de la Puerta de Transmisión [7], como la que se puede ver en la Figura 12.

Sin embargo, todas las estructuras discretas que encontramos necesitan una familia de transistores de efecto de campo en los que el canal no está unido internamente al sustrato, permitiendo cargar la capacidad puerta-canal sin afectar a la tensión del camino drenador-surtidor.

El principal problema de estos transistores es su elevado precio y muy poca variedad, siendo casi imposible encotrarlos. Además, generalmente se utilizan en aplicaciones de alta potencia por lo que su rendimiento para aplicaciones de baja señal suele ser bastante pobre.

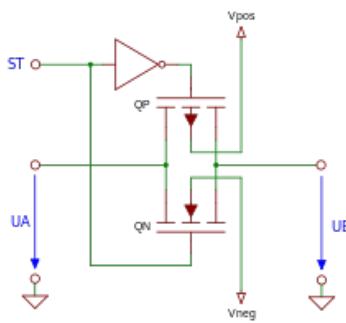


Figura 12: Puerta de transmisión con transistores con canal desconectado

Finalmente, descartamos la idea de utilizar componentes discretos y utilizamos una solución integrada. Por tanto, utilizamos el multiplexor analógico CD4053BC. [8]

Este multiplexor cuenta con tres canales en configuración SPDT, por lo que cada canal tiene un terminal en un extremo y dos en el otro. Este multiplexor cuenta con la ventaja de ser bidireccional, cosa de la que muchos otros carecen y es fundamental para nuestra funcionalidad.

Este multiplexor se utiliza para conectar las dos entradas de audio, que provienen de conectores Jack de audio de 3.5 mm a un GPIO que se conecta internamente a un ADC de la placa y para conectar

otro GPIO que se conecta internamente a un DAC a las entradas de los dos caminos de amplificación de audio.

2.3.4. Cambiador de nivel lógico

Un error que cometimos es no tener en cuenta la tensión de habilitación necesaria para commutar un canal del multiplexor, por lo que los 3.3V de salida de un GPIO no son suficientes para cambiar el canal del multiplexor. Esto provoca que se esté siempre seleccionado el canal correspondiente al nivel bajo.

Para solucionar esto, hemos construido un circuito cambiador de nivel lógico que adapta los 3.3 V de la placa a los 7 V necesarios para commutar el multiplexor (realmente el mínimo es aproximadamente 5V).

La solución que hemos pensado consiste en un inversor lógico TTL, que consiste en un transistor bipolar NPN con una resistencia de *Pull-up*. La única desventaja es la inversión de nivel, pero se corrige fácilmente en el software.

Se puede ver el diagrama de nuestra solución en la Figura 13, en la que se muestra un cambiador. Hemos soldado dos de ellos en una placa de prototipado, que se puede ver en la Figura 14.

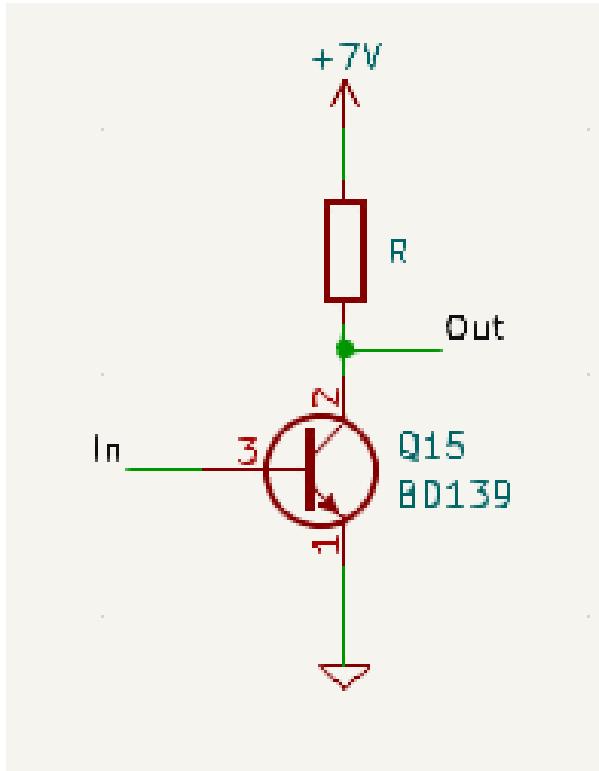


Figura 13: Circuito cambiador de niveles

2.3.5. Amplificador de audio para los auriculares

La salida del DAC de la placa es una señal entre 0 y 3.3 V con 12 bits de resolución. Por tanto, el audio que se genere tiene una componente continua que se debe eliminar.

Para eliminar esta componente continua hemos implementado una configuración de filtro paso alto mediante un filtrado pasivo y un seguidor de tensión realizado con un amplificador operacional. En el camino de realimentación del amplificador operacional se coloca una resistencia para anular la tensión de error de *offset* del amplificador operacional.

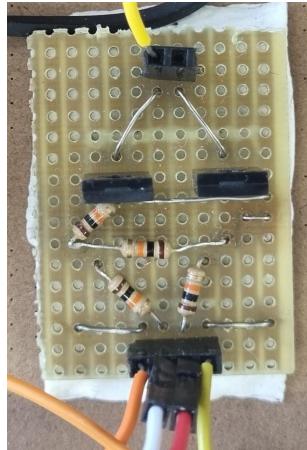


Figura 14: Foto del circuito cambiador de nivel

También se añade la misma estructura de transistores que en el circuito generador de tierra virtual, que ahora al estar también dentro del lazo de realimentación cuentan con la ventaja de que se anula la distorsión de cruce.

Los valores elegidos para los componentes del filtro son una resistencia de $100 \text{ k}\Omega$ y un condensador de 100 nF . Con ello, conseguimos una frecuencia de corte de:

$$f_c = \frac{1}{2\pi RC} \approx 16 \text{ Hz}$$

Elegimos este valor ya la banda de audición humana máxima es de 20 Hz a 20 kHz.

Se puede ver un diagrama de la solución que montamos en la Figura 15.

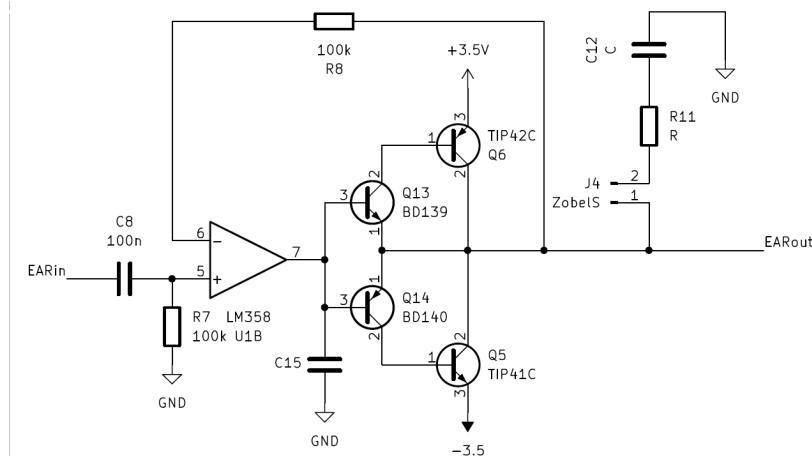


Figura 15: Circuito amplificador de auriculares

Sin embargo, al montar y probar el circuito detectamos que tenía un problema de inestabilidad para una frecuencia, lo cual es un problema común en los circuitos realimentados. Esto ocurre ya que la introducción de los transistores añade modificaciones impredecibles a la ganancia de lazo, provocando un muy molesto zumbido en los altavoces.

La solución que encontramos a este problema es la realimentación parcial mediante un condensador entre la salida del amplificador operacional y la base de los transistores. El tamaño del condensador afecta directamente a la reducción del ruido, cuanta más capacidad mejor lo elimina ya que hace una realimentación más directa. Sin embargo, cuanta mayor capacidad se le aplique, más se aprecia el efecto de la distorsión de cruce, la cual era eliminada al hacer la realimentación a la salida.

Experimentalmente probamos valores distintos determinando que el mejor balance entre ruido y distorsión de cruce se tiene para un valor aproximado de $33\mu F$. Sin embargo, este valor no es propio del circuito sino de las tolerancias y efectos parásitos de los componentes, por lo que podría variar mucho según las circunstancias.

Otra cuestión a la que nos enfrentamos es la cantidad de canales. Inicialmente diseñamos el circuito para ofrecer un solo canal de audio y dirigirlo a ambos canales, pero se pierde bastante calidad por lo que decidimos dejarlo en audio por un solo canal. Si se quisiera obtener audio por ambos canales o incluso estéreo, se debería duplicar este circuito y colocar uno por canal.

2.3.6. Amplificador de altavoces

El circuito amplificador de altavoces cuenta con el mismo paso bajo que el amplificador de los auriculares, pero se utiliza otra resistencia para aportar ganancia al circuito. Además, ya que se introduce la rama a tierra se añaden un par de condensadores para realizar un filtrado de alta frecuencia y eliminar el ruido debido a la tensión y corrientes de offset.

La frecuencia de corte del filtro paso bajo es de:

$$f_c = \frac{1}{2\pi RC} \approx 21,2 \text{ kHz}$$

Elegida igualmente para eliminar las frecuencias fuera del espectro auditivo humano.

La ganancia se elige para convertir el rango de salida ideal del DAC ($[0, 3,3] \text{ V}$) en el rango máximo ideal de los amplificadores antes de la saturación ($[0, 7] \text{ V}$) aunque la señal de audio no va a llenar el fondo de escala por su reducida amplitud. Por tanto, se elige una ganancia de tensión de:

$$A_v = 1 + \frac{R_4}{R_5} = 1,91V/V$$

Al igual que los otros dos circuitos, se introducen los transistores en el lazo de realimentación para la corriente, aunque en este caso no es necesario el condensador de realimentación parcial. Se tiene un esquemático de este subcircuito en la Figura 16.

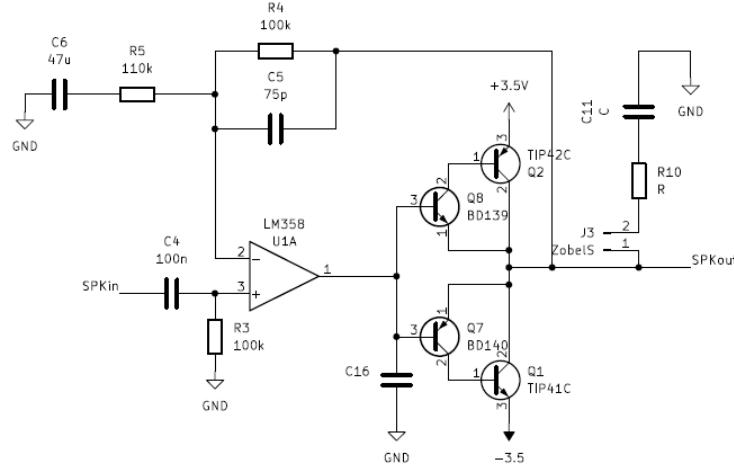


Figura 16: Circuito amplificador de altavoces

2.3.7. Diseño de PCB

Todos estos sistemas se han integrado en una única PCB para intentar maximizar la integridad de la señal de audio, lo cual se consigue totalmente si se conecta el circuito sin tener en cuenta el Apartado 2.7.

Hemos utilizado conectores Jack hembra de 3.5 mm para las dos entradas de audio y la salida a los auriculares. Un problema de este tipo de conectores es que el número de anillos puede variar en función

de si los auriculares tienen o no micrófono y si son mono o estéreo. Si se quiere utilizar unos auriculares con micrófono con nuestro sistema, se deben dejar ligeramente extraídos del conector para que haga mejor contacto la banda de tierra y mejorar significativamente el sonido.

La salida de altavoz se realiza a través de un terminal de dos tornillos para facilitar su conexión. Como ya hemos comentado anteriormente, el circuito de transistores cuenta con un condensador de estabilización que finalmente no hemos utilizado.

Además, hemos añadido la posibilidad de utilizar una red de Zobel, circuito que sirve para linealizar la respuesta en frecuencia de la inductancia intrínseca de los altavoces mediante un capacitor y una resistencia, pero finalmente no la hemos necesitado, por lo que tampoco está soldada.

El circuito cuenta además con un potenciómetro que sirve para ajustar el offset de la tensión de la tierra virtual gracias a los terminales específicos del OP07. Hemos utilizado componentes SMT para los componentes pasivos y los conectores de audio y THT para los circuitos integrados, transistores y conectores de terminal.

Se puede ver una imagen del circuito finalizado en la Figura 17 y el esquemático completo en el Apéndice C.

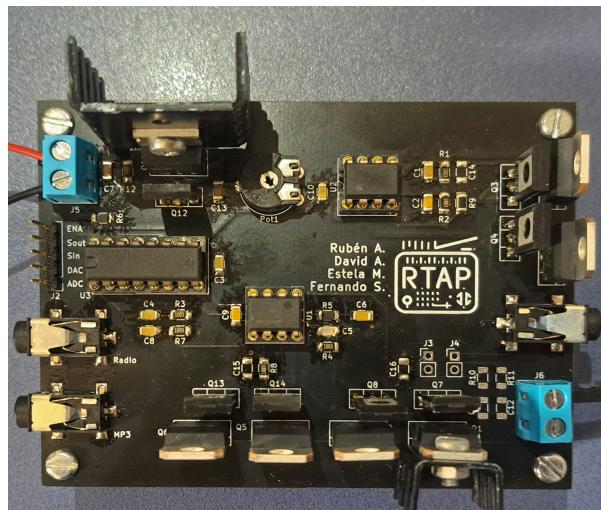


Figura 17: Circuito de audio completo

2.4. Módulo de radio

El modelo de radio elegido ha sido el Sintonizador FM RDA5807M, mostrado en la Figura 18, utilizado anteriormente en la asignatura de Sistemas Basados en Microprocesadores.

Dicho modelo se comunica con el microcontrolador mediante el protocolo I₂C. El sintonizador cuenta con un decodificador MPX, salida de audio stereo y un rango de sintonización de 87 MHz a 108 MHz debido a nuestra situación geográfica.

En cuanto a la señal de audio de salida, hemos encontrado que presenta una componente continua de 1.65V y una amplitud de aproximadamente 1V. Debido a la conectividad que presenta el sintonizador, nos hemos encontrado el problema de que, al unirlo junto al resto del proyecto, cortocircuita la señal de masa analógica con la señal de masa virtual creada en nuestro circuito. Dicho problema se comentará en el Apartado 2.7.

Por otra parte, necesita ser alimentado con una tensión entre 1.8V y 3.3V, soporta una corriente máxima de 10mA, cuenta con una relación señal-ruido típica de 55dB y presenta una distorsión armónica de audio total máxima del 0.2 %.

Todas las características de dicho sintonizador FM se han obtenido del datasheet ofrecido por el fabricante. [9]

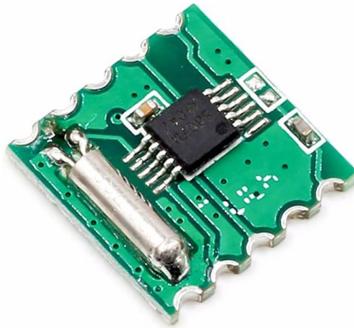


Figura 18: Sintonizador FM RDA5807M

2.5. Módulo MP3

El modelo de MP3 seleccionado ha sido el YX5300, el cual se muestra en la Figura 19.

Este reproductor se comunica con el microcontrolador mediante UART, con una velocidad de 9600 bps. También cuenta con una frecuencia de muestreo de 48 kHz y soporta tanto el formato MP3 como el formato WAV. Este modelo cuenta con un socket de tarjeta microSD en la cual se introducen las canciones, en los formatos antes mencionados, que se deseen reproducir. Dicha tarjeta deberá estar en formato Fat16 o Fat32 y tener como máximo 2 GB de almacenamiento.

En cuanto a la señal de audio obtenida a la salida del reproductor, podemos observar una salida bipolar entrada en 0V. Este comportamiento es totalmente inesperado ya que, al estar alimentado entre 3.3V y 0V, es extraño que la señal de salida pueda presentar valores negativos. Esto ha supuesto un gran problema en la unificación con el resto de módulos. Dicho problema y su solución será comentada en el Apartado 2.7.

Por otra parte, el reproductor debe ser alimentado con una tensión entre 3.2V y 5.2V, soporta una corriente máxima de 200mA y cuenta con un LED rojo que indica si está reproduciendo alguna canción.

Todas las características del reproductor MP3 se han obtenido del datasheet ofrecido por el fabricante. [10]

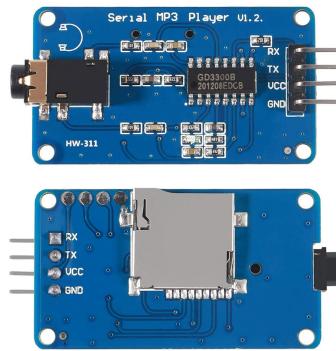


Figura 19: Reproductor MP3 YX5300

2.6. Módulo NFC

Para su desarrollo, hemos utilizado el periférico I2C1, el protocolo RF, la placa ANT7-T-M24SR64 [11] de ST Microelectronics y la aplicación móvil NFC Tools [12]. Además, a la hora de realizar comprobaciones desarrollando el módulo, hemos utilizado la app ST25 [13].

ANT7-T-M24SR64: La placa ANT7-T-M24SR64 es una placa que incluye un M24SR64-Y. M24SR64-Y es una tag dinámica NFC/RFID, EEPROM de interfaz dual, con protocolos RF e I2C. Se puede operar desde una interfaz I2C, un lector RFID o un teléfono con NFC.



Figura 20: Módulo NFC ANT7-T-M24SR64

Como se puede observar en la figura anterior, hay 6 pines:

- VCC: Alimentación 3.3 V.
- GND: Masa.
- SDA: Línea de datos del bus I2C.
- SCL: Señal de reloj del bus I2C.
- GPO: A nivel bajo, RF o I2C está siendo utilizado. A nivel alto, está libre.
- DIS: Activación/Desactivación de los comandos RF.

2.7. Entre Dos Tierras

El mayor problema al que nos hemos enfrentado en este proyecto es la gestión de las tierras en las señales de audio. Inicialmente diseñamos el circuito para que las señales de audio se conectaran entre la entrada de audio y la tierra virtual del circuito analógico, pensando que las señales serían compatibles con la entrada del ADC al estar alimentadas con los mismos niveles de tensión que los que las generan.

Sin embargo, descubrimos posteriormente que los circuitos compartían la tierra de audio de la entrada directamente con la salida, por lo que se realizaba un cortocircuito directo entre la tierra de la alimentación (o alimentación negativa visto desde el amplificador de audio) y la tierra virtual, por lo que se cortocircuitaban 3.5 V.

Esto era claro en el lado del MP3 ya que el cortocircuito era a través de un camino de baja impedancia y provocaba que se activara la protección, desactivando la salida de audio. Sin embargo, debido a la circuitería interna o a las conexiones de la radio, había una pequeña pero no mínima impedancia que provocaba un consumo muy elevado de corriente pero que no llegaba al amperio, por lo que la protección no se disparaba. Esto es muy peligroso ya que es una potencia que se está disipando en el interior del chip y probablemente provoque daños si se mantiene el circuito en dicha condición.

Para solucionar este problema, decidimos construir unos cables de sonido en los cuales solo se conecte el terminal que lleva la señal, deshaciendo la conexión que realizan los sensores internamente.

Con estos ajustes la radio funcionaba bien, sin demasiado problema (excepto el ruido del que hablaremos a continuación). Sin embargo, el MP3 presenta otro problema. Contraintuitivamente, a pesar de estar alimentado con una tensión unipolar, el circuito del MP3 consigue generar una tensión bipolar simétrica con la señal de audio, señal completamente incompatible con nuestro sistema de muestreo con un ADC de la placa.

Para mediar este problema, hemos montado otro circuito analógico accesorio que mediante un divisor de tensión simétrico y un condensador de desacoplo consigue añadir una componente continua de 1.65 V a la tensión simétrica de entrada, haciéndola completamente compatible con el sistema. Se añade una resistencia de *Pull-down* en la entrada para evitar picos de tensión en el encendido del sistema. Los valores de las resistencias pueden ser cualesquiera valores grandes siempre que sean iguales y el condensador interesa elegirlo lo más grande posible. El circuito está recogido en la Figura 21.

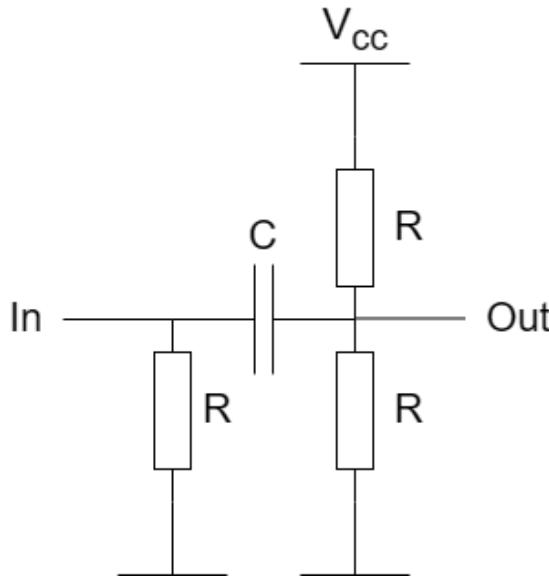


Figura 21: Esquemático del circuito sumador

Se ha montado el circuito sobre una placa de baquelita, como se puede ver en la imagen de la Figura 22.



Figura 22: Foto del circuito sumador analógico

Hemos elegido un valor de $100\text{ k}\Omega$ para las resistencias y aproximadamente $1\text{ }\mu\text{F}$ para el condensador, lo cual nos ofrece buenos resultados. Sin embargo, primero realizamos las pruebas con un condensador cerámico y no conseguimos que la tensión se estabilizara, por lo que se desplazaba el valor lentamente hacia uno de los rafles de alimentación. La solución que encontramos fue sustituirlo por un condensador de tántalo, que si bien tiene un tamaño físico considerablemente superior, consigue mantener de manera muy estable la tensión del circuito.

Este circuito funciona correctamente pero tiene la desventaja de decrementar ligeramente la tensión de entrada, provocando una caída en la relación señal-ruido del sistema.

Una vez solucionado el problema de las masas, aparece un ruido muy elevado en forma de zumbido y ruido blanco, por lo que el audio es de bastante baja relación calidad ruido. Esto es principalmente debido a la longitud de los conductores por lo que va la señal, la cantidad de circuitos que atraviesa, etcétera.

Además, la desconexión de las masas de los cables de audio provoca que el camino de retorno de las señales tenga que atravesar el resto del circuito y se deja de tratar la señal como un par diferencial, por lo que se pierde bastante calidad debido a la diferencia de tensión en las masas y algún posible bucle de masa al que se le acople algún ruido.

Por todo ello, si se quiere disfrutar de la máxima calidad de audio que puede ofrecer nuestro circuito, se

debe desconectar la masa de la placa de la del circuito de amplificación de audio. El principal inconveniente de ello es que se deja de poder gestionar el multiplexor y la habilitación a través de los GPIO y no se puede realizar el procesado digital de señales.

Si se conectan los pines de selección y habilitación a los raíles de alimentación para seleccionar la configuración y se conecta un jumper entre los pines de ADC y DAC, se tiene una buenísima calidad de sonido, tanto en el altavoz como en los cascos. Se puede igualmente controlar la radio y el MP3 mediante las interfaces ya que eso no depende de la masa del circuito.

La mejor solución a este problema sería la realización de un circuito con alimentación simétrica verdadera. Esto se podría conseguir mediante dos baterías en serie (aunque sería un desperdicio ya que una solo se utilizaría para la parte negativa de la señal y no alimentaría la placa) o mediante la generación de una tensión negativa con un convertidor, por ejemplo, de tipo reductor-elevador con topología inversora. Igualmente se tendría que tener en cuenta la naturaleza bipolar de la señal del MP3 y se debería incluir el circuito de *offset* a la placa o utilizar un cambiador de nivel analógico.

2.8. Conexión de la alimentación a la placa

Otro problema significativo que hemos encontrado es la conexión de la placa a la alimentación por baterías. La placa STM32F769NI-Disco indica que se puede alimentar con una tensión de entre 7 y 12 voltios en el pin de Vin siempre que se seleccione ext5V en los jumpers de selección de alimentación.

Probamos a alimentarlo con una fuente de tensión de laboratorio y el circuito funcionaba adecuadamente, consumiendo aproximadamente 300 mA. Sin embargo, al alimentarlo desde nuestro subsistema analógico, la placa funciona adecuadamente durante aproximadamente cinco segundos para después quedarse congelado. Hemos podido comprobar que es únicamente la CPU que se congela ya que las DMAs de audio siguen funcionando, reproduciendo el último contenido del buffer en bucle.

Hemos comprobado que no es un problema de tensión ya que, aunque en la hoja de catálogo indique que la placa requiere de mínimo 7 voltios, funciona bien incluso con 6,5 V de la fuente de alimentación.

Tampoco es un problema de corriente ya que, como se explica en el apartado de test hardware, la placa puede aportar mucha más corriente que el aproximadamente medio amperio que requiere la placa.

3. Software

3.1. Interfaz de usuario

Hemos desarrollado dos interfaces de usuario, una sobre una web y otra sobre una pantalla táctil.

3.1.1. Interfaz web

Para la interacción con el sistema, hemos diseñado un servidor web mediante el lenguaje HTML, estilizado mediante CSS y con algunas funciones creadas mediante JavaScript.

El procesamiento de datos recibidos desde el servidor web y la creación dinámica de datos para la web se generan en diferentes archivos CGI, los cuales se explicarán en el Subapartado 3.3.11.

El servidor web cuenta con 4 páginas web diferentes, una principal, una para gestionar la radio, otra para gestionar el reproductor MP3 y otra para gestionar el procesado de audio. En todas estas páginas, en la esquina superior derecha, se encuentra la fecha y la hora del sistema. Todas estas páginas se explican a continuación.

3.1.1.1 Configuración General

Esta página web cuenta con una sección llamada *Camino de Audio*, la cual cuenta con 4 botones que nos permiten seleccionar tanto la entrada, Radio o MP3, tanto la salida de audio, Auriculares o Altavoz.

También cuenta con una sección llamada *Bajo Consumo* que cuenta con tan solo un botón que nos permitirá poner el microcontrolador en modo bajo consumo.

Por último, cuenta con una sección llamada *Consumo*, que contiene un widget, el cual se ha obtenido de [14] que nos permite visualizar de forma dinámica el consumo medido en el sistema.

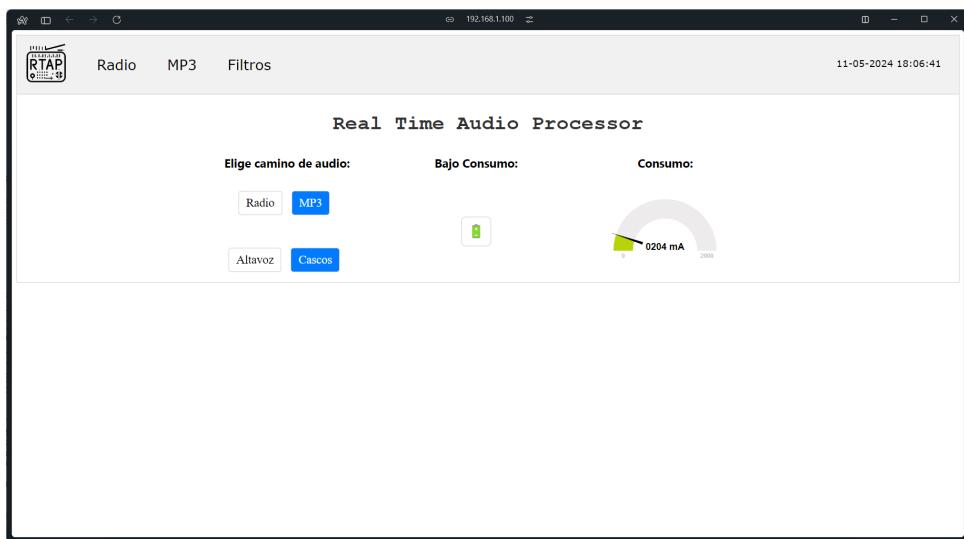


Figura 23: Página Principal

3.1.1.2 Página Radio

Esta página web cuenta con una primera sección llamada *Sintonizar una frecuencia* la cual nos permite introducir la frecuencia que deseemos sintonizar en el recuadro blanco. Luego, mediante el botón Sintonizar, podremos sintonizar dicha frecuencia en el Sintonizador FM.

A continuación, se encuentra una sección llamada *Seek*, en la cual se encuentran dos botones. El primero, que contiene una flecha hacia arriba, nos permite realizar un *SeekUp*, es decir, sintonizar una frecuencia mayor con más potencia que un determinado umbral. De forma análoga, el otro botón, ilustrado mediante una flecha hacia abajo, nos permite realizar un *SeekDown*, es decir, sintonizar una frecuencia menor, pero que tenga más potencia que dicho determinado umbral.

La siguiente sección llamada *Volumen*, contiene un slider horizontal que nos permite seleccionar el volumen de la señal de audio. También encontramos un botón de mute, es decir, situar el volumen a 0.

Por último, encontramos la sección *Salida*, la cual cuenta con dos botones que nos permiten seleccionar la salida de audio deseada entre Altavoz o Auriculares.

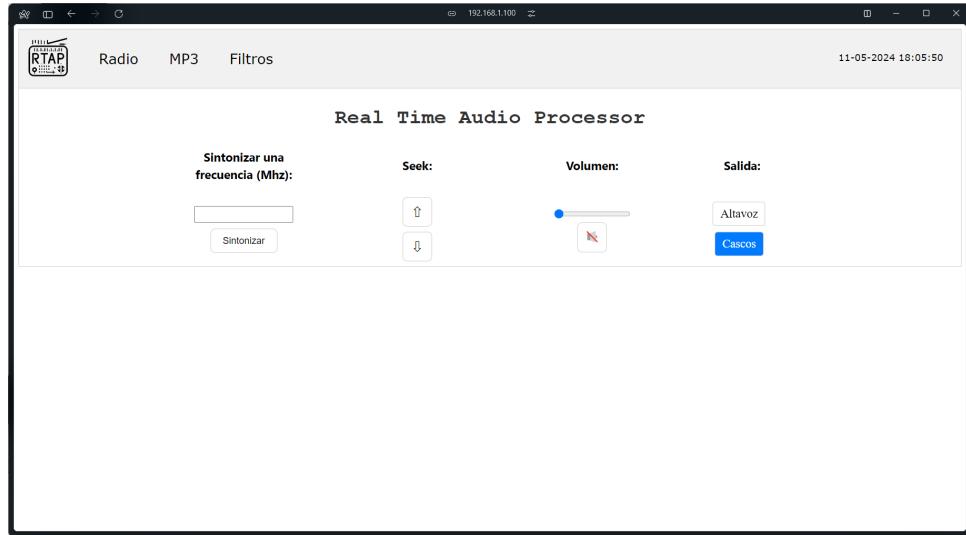


Figura 24: Página Radio

3.1.1.3 Página MP3

En primer lugar, nos encontramos con una sección llamada *Canciones*, la cual cuenta con un menú desplegable con lista, con sus correspondientes nombres, de las posibles canciones. Junto a dicho menú, se cuenta un botón que nos permite confirmar la canción seleccionada.

A continuación, encontramos la sección denominada *Control* la cual cuenta con 4 botones. El primero, nos permite seleccionar la canción anterior a la canción actual. A continuación, nos encontramos con un botón que nos permite tanto pausar como continuar la reproducción de la canción actual. El siguiente botón nos permite seleccionar la siguiente canción de la lista. Por último, el botón de abajo, nos permite activar y desactivar la puesta en bucle de la canción actual.

De forma análoga a la página de la Radio, las siguientes dos secciones nos permiten modificar el volumen del sistema y la salida de la señal de audio.

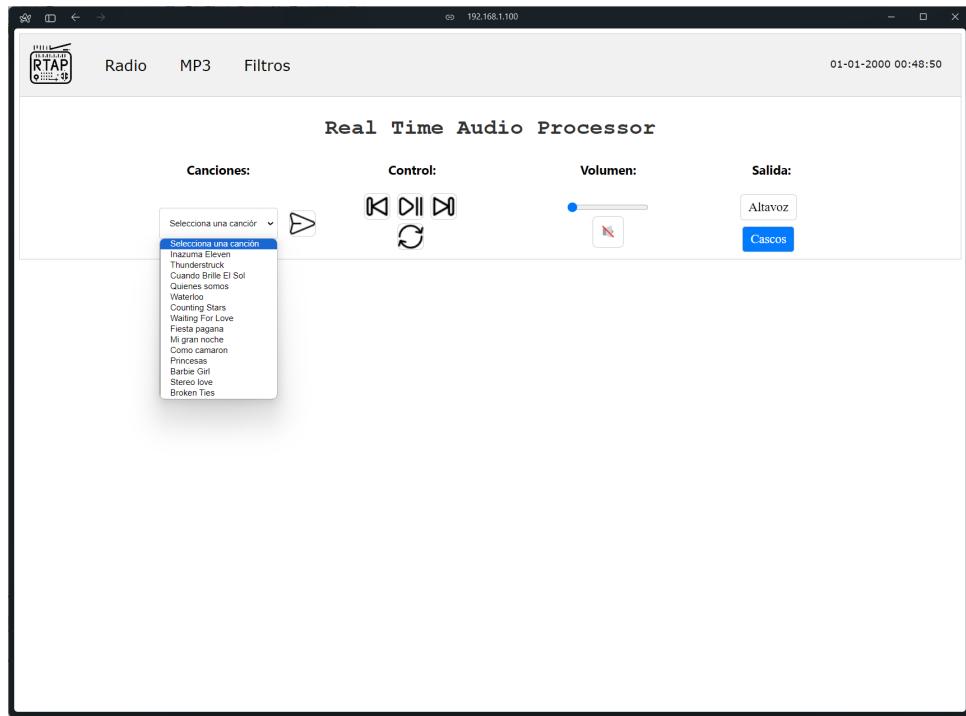


Figura 25: Página MP3

3.1.1.4 Procesamiento de Audio

La última página web se encarga de todo el procesamiento de audio. En primer lugar, nos encontramos con la sección llamada *Ecualizador*, el cual nos permite elegir, mediante unos sliders verticales, entre un rango de valores, la ecualización que se deseé aplicar en las diferentes bandas posibles.

A continuación, en la sección denominada *Guardar Conf.*, nos encontramos un botón que nos permite guardar la configuración de los distintos filtros en la tarjeta microSD conectada al sistema.

Por último, de la misma manera que en las dos anteriores páginas, nos encontramos los controles que nos permiten modificar el volumen del sistema y seleccionar la salida de audio.

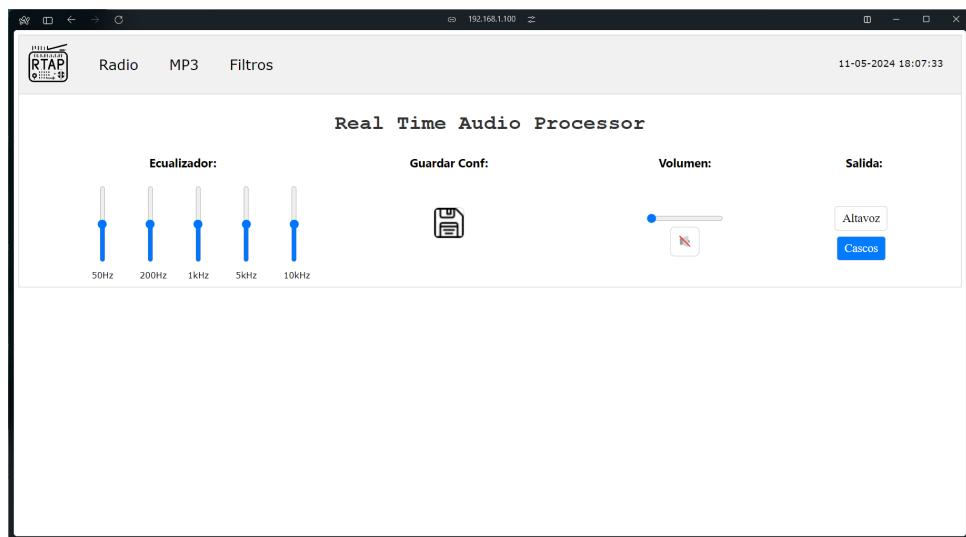


Figura 26: Página Filtros

3.2. Interfaz táctil

Una vez se ha configurado una función de bajo nivel para representar los píxeles en la pantalla, que se explica en el apartado 3.3.1.3, se pueden utilizar las abstracciones de alto nivel que ofrece LVGL.

La interfaz gráfica de RTAP consta de 4 pestañas: Inicio, Radio, MP3 y Filtros. A su vez, cada pestaña dispone de un lienzo en color gris claro, sobre el que se añaden paneles, con un fondo blanco y un borde gris de 2 píxeles de grosor. Cada panel implementa una funcionalidad, o muestra la información relevante. Todas las pestañas comparten una filosofía: Los paneles principales ocupan la totalidad de la pantalla, haciendo la interfaz muy intuitiva, y, si se desliza el panel hacia la parte superior, se muestran los créditos del proyecto: El título, la asignatura y los estudiantes involucrados. Además, todo el proyecto utiliza la fuente *Montserrat*, variando el tamaño según la importancia del contenido. A continuación, se explica cada una de las pestañas, excluyendo el panel de créditos.

3.2.1. Inicio

La pestaña de inicio consta de dos paneles principales: Configuración rápida y Consumo.

- **Configuración rápida:** Este panel está formado por tres grupos de botones. El primer y el segundo grupo se emplean para seleccionar la salida y la entrada, respectivamente. Como el sistema no permite la opción de seleccionar las dos entradas o las dos salidas simultáneamente, estos botones son excluyentes: marcar uno desmarca el otro. Como estilo, se ha optado por un color azul claro para representar la opción seleccionada, y un color más oscuro para la no seleccionada. Por último, este panel tiene un botón que hace que el sistema entre en modo de bajo consumo. Los botones se distribuyen según una rejilla fija, y los grupos de estos se dividen mediante un espaciado uniforme, calculado de manera dinámica por LVGL.
- **Consumo:** Este panel está formado por una escala en forma de arco, que ocupa 225º de la circunferencia. Además, se le aplica una rotación de 180º para conseguir la posición deseada, que deja espacio suficiente para colocar una etiqueta de texto.

La escala, que se utiliza para representar de forma gráfica el consumo del sistema, admite valores entre 0 y 1500, que se interpretan como miliamperios. A la escala se le aplican tres formatos diferentes: El primero es un color verde, para los valores entre 0 y 375, el segundo, un color azul, entre 375 y 1125, y el tercero, en rojo, para los valores más altos, entre 1125 y 1500. En total, la escala cuenta con 25 ticks, y cada 4 de estos, se introduce uno principal, es decir, para los valores más significativos, de 0 A, 0.25 A, 0.5 A, 0.75 A, 1 A , 1.25 A y 1.5 A. Estos ticks principales tienen un grosor de 2 píxeles, mientras que los secundarios tienen un grosor de 1 píxel. LVGL calcula la longitud de cada tick de manera dinámica. Por último, la escala cuenta con una aguja, en color rojo y con una longitud máxima de 80 píxeles, que apunta al valor del consumo instantáneo, que además se representa en la etiqueta de texto.

El panel se organiza según una rejilla flexible organizada en forma de columna. Esto significa que simplemente se declaran los objetos y se les da un tamaño, y LVGL los coloca de forma dinámica apilados verticalmente. Sin embargo, para conseguir que la etiqueta se muestre en el lugar apropiado, se le da una posición fija relativa al panel.



Figura 27: Pantalla principal de la aplicación

3.2.2. Radio

La pestaña de radio está formada por tres paneles principales: Uno para la radio, otro para el volumen, y otro para la salida.

- **Radio:** Este panel es el más grande de esta pestaña, ocupando dos terceras partes del espacio. Este panel es muy complejo, estando formado por una superposición de elementos. De manera simple: En la parte posterior encontramos un título, que indica que el panel es para sintonizar una frecuencia. En la posición inmediatamente inferior, se muestra un número, que en caso de no tener ningún valor, se mostrará en color sombreado. Hay varias formas de modificar este valor, que se detallan más adelante. Después encontramos una escala, que simula la apariencia de una radio clásica, y cuyos valores, entre 87 y 108, se interpretan como megahercios. Sin embargo, en un primer plano, encima de la escala, encontramos un slider, del cual solo se representa el mando, en color rojo y con un borde negro. En LVGL un slider solo puede tener valores enteros, por lo que este toma valores entre 870 y 1080, para mantener un decimal, interpretándose el valor como centenas de kilohercio. En la parte inferior hay un espacio en blanco, cuya utilidad se comenta a continuación, y después hay una fila con un objeto en forma de caja, inicialmente vacío, y a su derecha, tres botones.

Empezando por el slider, que puede ser la opción más intuitiva, un deslizamiento sobre la escala tendrá diferentes consecuencias: El texto que tiene encima se modificará, acompañando el valor del texto, y, en caso de ser una cadena conocida, en el espacio en blanco que se mencionó anteriormente, se representará el texto correspondiente al nombre de la cadena. Sin embargo, para no saturar la cola de mensajes ni a la radio, la cadena marcada sólo se sintoniza una vez se ha soltado el slider.

Otra posible forma de modificar el valor de la frecuencia es mediante los botones de *Seek*. El sistema busca la siguiente cadena con una señal aceptable, la sintoniza, y modifica el valor del texto y del slider. Otra opción es tocar el valor del texto, y se desplegará un teclado numérico, que permite sintonizar de manera rápida y precisa una cadena, solo en caso de haber introducido un valor correcto. En caso de que el valor introducido no sea un valor posible, o sea un número mal formado, el slider se truncará hacia el valor correcto más próximo. Para cerrar el teclado, se puede utilizar cualquiera de los dos botones pensados para ello, o tocar en cualquier parte de la pantalla.

Además, se dispone de un botón de favoritos, que almacenará en la caja todas las cadenas que se quieran guardar (no es persistente). En caso de que se conozca el nombre de la cadena, será esto lo que se muestre en la lista, en caso contrario, se mostrará únicamente la frecuencia. Para recuperar cualquier cadena guardada, solo habrá que buscarla en la lista.

Por último, destacar que cualquier cambio en la web se verá reflejado de forma inmediata en el slider y en todos los cuadros de texto.

- **Volumen:** Este panel permite ajustar el volumen global del sistema. Se mantiene sincronizado con los paneles de volumen de otras pestañas, y presenta un botón de mute, que ejecuta una animación,

variando el valor hasta 0 de forma suave, o, en caso de estar ya en cero, aumentándolo hasta la posición previa. Igual que con otros sliders, para no saturar los mecanismos de sincronización, el valor solo es enviado una vez se ha soltado.

- **Salida:** Este panel también permite seleccionar la salida del sistema. Se mantiene sincronizado con los botones que ofrecen la misma función en otras pestañas, y mantiene el mismo estilo que el de la pestaña de inicio para indicar cuál es la opción seleccionada.



Figura 28: Pantalla de la radio

3.2.3. MP3

La pestaña del MP3 mantiene muchas similitudes con la de la radio: Se compone de tres paneles, de los cuales el principal también ocupa dos tercios de la pantalla, y los dos paneles de su derecha son idénticos a los explicados anteriormente.

El panel principal del MP3 también es un panel muy complejo. En este caso, el panel a su vez está compuesto por varios paneles. En un primer momento, se aprecian los 3 botones principales del MP3: anterior, play/pause y siguiente. El color de estos botones se ha pensado para romper con la monotonía del sistema, dando una impresión más alegre. El degradado se calcula automáticamente por LVGL, no se aplica ninguna textura que ocupe memoria.

En la parte inferior del panel hay una pestaña, que si se desliza hacia arriba, pasa a ocupar el 70 % del subpanel. Sin embargo, a los botones se les permite ocupar el 40 % del panel, con esto se logra la impresión de que los botones no terminan de esconderse nunca. Esta pestaña representa la lista de canciones que se tiene almacenada en la tarjeta SD. Al hacer click sobre una canción, esta se envía al MP3, y además, se reproduce una animación, que sustituye durante unos segundos al título de la pestaña, representando la canción. Nuevamente, si se oculta la lista de canciones, los botones de control pasan a ocupar el 100 % de su panel.



Figura 29: Pantalla del MP3

3.2.4. Filtros

La pestaña correspondiente a los filtros muestra las opciones de ecualización del sistema, además de algunos ajustes para el audio. En concreto, esta pestaña consta de 4 paneles:

- **Ecualizador:** Es el panel principal de la pestaña, y ocupa la mitad de esta. Se compone de 5 sliders, que manejan cada uno una banda. Como en sliders anteriores, el valor únicamente se envía cuando se libera el slider. Modificar un valor en la página web ejecuta una animación para sincronizar el valor.
- **Volumen:** Tiene un tamaño ligeramente inferior al de las pestañas anteriores, pero mantiene el mismo valor y funcionalidad.
- **Guardar config:** Es un pequeño panel con un único botón, para almacenar la configuración.
- **Configuración de audio:** Es parecido al panel de configuración rápida de la pestaña de inicio, manteniendo el estilo y todos los botones, excepto el de bajo consumo, que en este caso es reemplazado por un botón para restablecer el valor de los filtros. Pulsar el botón hace que se ejecuten 5 animaciones, una para cada slider, poniendo su valor a cero de forma suave.



Figura 30: Pantalla de los filtros

3.3. Módulos software

Para la implementación software se han diseñado módulos independientes para cada tarea, implementando casi todos como un hilo (o tarea) del sistema operativo de tiempo real de Keil, utilizado a través

de la API CMSIS RTOS2.

Para la comunicación entre hilos hemos utilizado principalmente colas de mensajes al ser la opción más cómoda y sencilla de implementar, aunque hay un par de candados de exclusión mutua (`mutex`) y alguna `flag` de hilo.

Se puede ver un diagrama de bloques software en la siguiente figura:

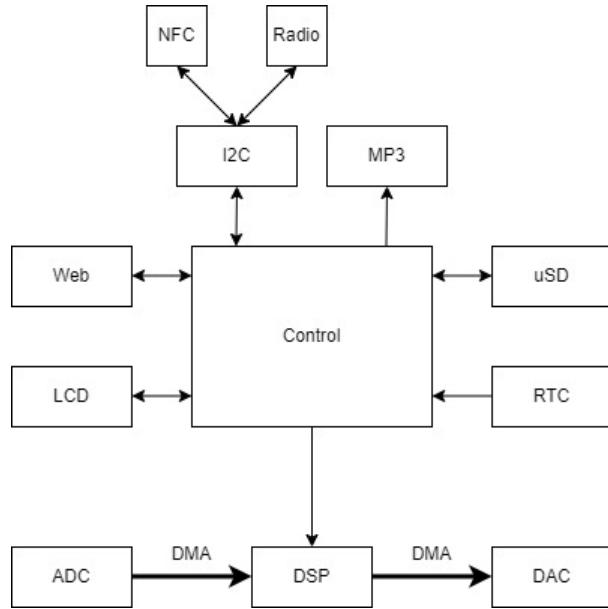


Figura 31: Diagrama de bloques software del proyecto

3.3.1. Módulo de LCD

3.3.1.1 La parte técnica

La placa STM32F769-disco que se ha utilizado en este proyecto dispone de una pantalla LCD táctil de 4.3”, con una resolución de 800 x 480 píxeles y una profundidad del color de 16 bit. Esto permite crear interfaces gráficas atractivas, y, gracias al uso de las DMA, con un coste computacional asumible. Por tanto, después de estudiar las diferentes opciones disponibles, se ha diseñado una interfaz completa, que permite el control completo del sistema.

3.3.1.2 Software y librerías disponibles

A la hora de crear una interfaz gráfica, la opción más lógica es utilizar una librería de más alto nivel, o un software de creación de interfaces, para abstraer el manejo de cada píxel individual, pero, en el mundo de los microcontroladores, estas opciones son bastante limitadas. Algunas de las opciones que se han estudiado son:

- **Embedded Wizard:** Este software permite la creación de interfaces gráficas de una manera aparentemente sencilla, pero es código cerrado, y para utilizarlo de forma gratuita hay que asumir una marca de agua con su logo. Además, tienen su propio sistema operativo, que si bien no es muy distinto de las opciones conocidas, maximiza el riesgo de fallo a la hora de, por ejemplo, implementar el servidor web. Por tanto, esta opción se descartó.
- **EmWin:** Este software es la herramienta de Keil para la creación de interfaces, y viene incluida como *software pack* dentro del programa. Incluye una función de generación de interfaces *drag and drop*, lo que significa que, desde su programa, solo hay que colocar los elementos que se quieran tener en la interfaz en su sitio adecuado, todo desde una interfaz gráfica, abstrayendo el código. Sin embargo, las opciones de este software son muy limitadas, y las interfaces que genera tienen un aspecto rudimentario y obsoleto, por lo que esta opción también se descartó.

- **TouchGFX:** Este software, propiedad de STM, presenta una interfaz gráfica para la generación automática de código. Es un software muy potente, con el que es fácil generar interfaces modernas y visualmente atractivas. Además, incluye numerosos ejemplos, tanto en su aplicación, como en CubeMX. En un principio, se seleccionó esta opción, pero presenta el problema de ser de código cerrado, y es muy difícil adaptar el código que se genera para que sea compatible con Keil. Por tanto, finalmente se descartó.
- **LVGL:** Little Versatile Graphic Library es una librería de código abierto ampliamente utilizada en el mundo profesional. Varias empresas multinacionales, como Xiaomi o LG, utilizan actualmente adaptaciones de esta librería en algunos de sus productos. Es cierto que esta opción no dispone aún de herramientas para la generación automática de código, pero la librería es relativamente fácil de manejar. Con LVGL se pueden generar interfaces de todo tipo, y para el proyecto se ha seleccionado por su versatilidad y su fácil manejo. Además, ahora está presente en Keil como *Software Pack*, por lo que su integración es absoluta. En nuestro proyecto, se utiliza la versión 9.1, que en el momento de redacción e implementación del proyecto, es la última versión estable de la librería.

3.3.1.3 El bajo nivel

LVGL es una librería de alto nivel, que es capaz de dibujar formas sobre un *array*. Sin embargo, es responsabilidad de la persona que implementa la librería, el representar este *array* en la pantalla. De este modo, el programador genera una función que representa un conjunto de píxeles en su pantalla, y LVGL se encarga de llamar a esta función cuando corresponda. Es decir, el tiempo de refresco de la pantalla variará según las necesidades del momento, liberando el uso de CPU cuando la pantalla tiene una imagen estática.

La primera aproximación posible para esta función es un simple bucle: itera todo el *array* de píxeles y los envía a la pantalla. Sin embargo, esta aproximación es muy intensiva en el uso de CPU, además de poco eficiente. Por ello, una vez se ha determinado que el funcionamiento de la pantalla y el de LVGL es correcto, conviene modificarla. En el caso de RTAP, esta función utiliza la DMA 2, el *Stream 2* y el Canal 2, en configuración *Memory To Memory*, consiguiendo alcanzar la tasa de refresco máxima que soporta la pantalla, de 30 fps, con un uso de la CPU mínimo, excepto cuando se ejecutan animaciones, que el uso de CPU aumenta considerablemente. Toda la configuración de bajo nivel reside en los ficheros *tft.c*, *tft.h*, *touchscreen.c*, *touchscreen.h*. Estos se pueden consultar en el Apéndice E.

Una de las razones por las que la tasa de refresco disminuye es por el modo de funcionamiento de LVGL: admite diferentes configuraciones de *buffer* que pasará como parámetro a la función de bajo nivel. En nuestro proyecto se utilizan de manera simultánea dos *buffers* (la pantalla lee de uno mientras el controlador escribe en otro, y luego conmutan), pero cada uno solamente es del tamaño de la décima parte de la pantalla. Gracias a esta configuración es posible ahorrar mucha memoria, a cambio de un mayor uso de la CPU.

3.3.1.4 Consideraciones sobre la pantalla y el bajo consumo

La pantalla de la placa es un periférico complejo, que, como se ha explicado, utiliza la DMA para representar una imagen. Pero, cuando se entra en modo bajo consumo, la pantalla accede constantemente a los mismos datos: el reloj del procesador está parado, por lo que la información de los píxeles no se actualiza. Esto tiene un efecto indeseado, que es que la pantalla no se apaga, se queda con la última imagen que se cargó antes de entrar en modo de bajo consumo, lo que implica que el consumo es mayor del deseado. Por tanto hay que tomar la precaución de apagar la pantalla antes de entrar en este modo. RTAP implementa esta funcionalidad de manera transparente para el usuario: la aplicación apaga la pantalla cuando entra en modo de bajo consumo en cualquier caso, independientemente de que la orden se envíe desde el botón destinado a ello en la interfaz táctil o en la interfaz web.

3.3.1.5 Consideraciones sobre LVGL

Sobre la memoria

Después de haber repasado de forma general la interfaz del proyecto, se entiende que para representar un sistema tan complejo hace falta una cantidad considerable de memoria. En concreto, se asignan 131072 bytes de memoria a LVGL, y un stack de 12400 bytes al hilo que gestiona el funcionamiento de LVGL. La cantidad de memoria empleada es más que suficiente para el correcto funcionamiento de la placa, se deja un margen de en torno al 30 %.

Para el proyecto de RTAP es necesario ser generosos con la cantidad de memoria de LVGL, porque, si bien es cierto que solo se mantiene en memoria los objetos que se están representando en el momento, hay diferentes elementos cuya cantidad de memoria variará entre dos usos de la aplicación. En concreto, estos elementos son: la lista de emisoras guardadas de la radio, y la lista de canciones del MP3. No se puede determinar con antelación el número de emisoras que almacenará el usuario, o el número de canciones que se deben representar. La configuración empleada está pensada para soportar cualquier caso de uso.

Sobre los objetos

En LVGL un objeto es un *struct* con información sobre este. Los objetos en LVGL son extremadamente genéricos, hasta el punto de que, para la librería, un panel contenedor de botones es del mismo tipo que los propios botones, que a su vez son del mismo tipo que un teclado, que un *slider* o que una escala. Esto implica una dificultad, y es que es muy sencillo pasar como parámetro un objeto equivocado a una función, con el correspondiente efecto indeseado, que suele ser un *Hard Fault*, pero, también ofrece la ventaja del concepto de jerarquía: Independientemente del tipo de objeto, será hijo de otro objeto. El puntero al objeto padre se incluye en el *struct* que define un objeto, además de punteros a todos los objetos hijos. El único tipo de objeto que no tiene padre es especial, y es la propia pantalla. Se pueden definir varias pantallas, pero en RTAP se trabaja únicamente con una, de modo que cualquier objeto es accesible para los demás a través de su árbol (esto evita el uso de variables globales).

Sobre los eventos

LVGL tiene diferentes métodos de notificación de eventos, muy útiles en un sistema orientado a eventos, como es una interfaz gráfica. Hay tres grandes grupos de eventos en nuestro proyecto: los generados por la pantalla, los generados por la web, y los generados periódicamente por el programa. En LVGL, un evento no es más que un *struct* con la información pertinente, que incluye el objeto que lo provocó e información relevante, como el tipo de evento.

En cuanto a los primeros, desde la interfaz gráfica se asignan funciones de *callback* a los objetos que las deben de generar, como los botones o *sliders*. Como en LVGL un evento contiene la información sobre el tipo de evento, se pueden tomar numerosas decisiones. Por ejemplo, RTAP solo notifica de que un *slider* ha variado su valor una vez se ha soltado, para no saturar a las colas de sincronización, pero, en el caso del *slider* de la frecuencia de la radio, sí que se toma una acción mientras se está variando la posición de este: se representa el nombre de la cadena en el lugar específico. Por otro lado, desde la interfaz gráfica se pueden tomar acciones que afectarán a otros objetos del sistema: el botón de *mute* (de cualquier pestaña) ejecuta una animación en el *slider* del volumen, y el botón de restablecimiento de los valores de los filtros ejecuta animaciones en los *sliders* de estos.

Todos los eventos se notifican a la web mediante elementos de sincronización, explicados en Subapartado 3.3.2. Cabe destacar que, desde el hilo que gestiona la pantalla y desde las diferentes funciones de *callback*, no se toman acciones que afectan al sistema en general, únicamente se envían mensajes de control, para que un hilo controlador ejecute las acciones pertinentes.

Los eventos generados por la web presentan una dificultad: al ser la web un elemento invisible para LVGL, no se puede asignar una función de *callback* directamente. Afortunadamente, la librería implementa métodos de notificación de eventos generados externamente, pero se debe de tener en cuenta otra consideración: el evento se notificará en el próximo refresco de la pantalla. Para cuidar la calidad del audio y evitar cortes, RTAP incluye un mecanismo que limita la tasa máxima de refresco de la pantalla. De este modo, cuando se está ejecutando una animación, por ejemplo, un cambio de pestaña, LVGL no ocupa toda la CPU, evitando así cortes molestos en el sonido. Pero esto implica que, como máximo, LVGL será notificado de un evento a los 15 milisegundos, siendo más común una notificación cada 33 milisegundos (30 refrescos por segundo). Si se genera más de un evento en este tiempo, se perderá. Por ello es importante que los *sliders* de la web envíen también su valor únicamente cuando el usuario ha soltado el mismo, de otro modo, el comportamiento es impredecible.

El último grupo de eventos es para los que no se provocan ni por la web ni por la pantalla, aunque, para

LVGL, este tipo es exactamente igual que los de la web. Simplemente es algo que LVGL no puede predecir ni gestionar. En nuestro caso, el consumo entra dentro de este grupo, cuyo valor se mide cada segundo, y se notifica al hilo controlador de la pantalla, que a su vez lo representa en la interfaz, concretamente en la pestaña de inicio, como se explica en Apartado 3.2.

Sobre la configuración de LVGL

En este punto se describen las acciones necesarias para hacer funcionar LVGL en la placa, y poder crear así interfaces gráficas modernas y visuales.

LVGL está disponible como *Software Pack* de Keil, y, desde la versión 9, se han cambiado la forma de integrar sus módulos, consiguiendo una organización muy similar a la de los demás *Software Packs*. Esto consigue que la configuración sea bastante sencilla. Tan solo se debe seleccionar la versión adecuada, que en nuestro caso es la 9.1, por ser esta la última versión estable en el momento de la redacción del documento, y seleccionar los paquetes *Display* y *Essential*. Además, existen numerosos módulos adicionales que se pueden seleccionar, que son adicionales al núcleo de LVGL y cuyo propósito es complementar a este. Desde RTAP no utilizamos ninguno de estos módulos adicionales, puesto que están pensadas para que LVGL sea el núcleo del proyecto, que es distinto de nuestro caso de uso, pero hemos creado numerosas adaptaciones similares específicas para nuestro proyecto.

Uno de los archivos más importantes a la hora de trabajar con LVGL es `lv.conf.h`. En este fichero se definen las principales opciones de LVGL, como la memoria que se le asigna, las fuentes que utiliza la interfaz, o el soporte para sistemas operativos en tiempo real. La versión que se incluye por defecto tras seleccionar los *Software Packs* es suficiente para que LVGL funcione, pese a que en proyectos complejos como RTAP pueda ser necesario aumentar la memoria del sistema, como se explica a continuación.

3.3.2. Módulo de control

El módulo de control es el encargado de orquestar la aplicación. Para incrementar el rendimiento, se ha decidido modelar con una arquitectura basada en eventos, por lo que el módulo se encuentra permanentemente esperando en una cola y sólamente reacciona cuando le llega un mensaje por ella, volviendo a esperar al finalizar la respuesta.

Para evitar tener que realizar una estructura de polling de colas, en la que el hilo debe comprobar constantemente varias colas, se ha implementado una estructura unificada de mensajes de entrada al módulo de control, de forma que solo se tiene que esperar a una sola cola. Además, cuenta con acceso a las colas de entrada de todos los demás módulos, en las que introduce mensajes como reacción a los eventos que ocurran.

Se podría realizar una estructura que contuviera todos los posibles parámetros de mensajes y un identificador para saber cuales son los útiles, pero esto es muy poco óptimo en memoria y puede tener un gran impacto en el rendimiento de la aplicación. Por ello, hemos decidido implementar una estructura basada en uniones con un campo que identifica cual es la forma de dicha unión.

Fragmento 1: Estructura para los mensajes al control

```

/**
 * @brief Enumeracion de los tipos de mensaje de entrada al modulo de control
 */
typedef enum {
    MSG_NFC,    /**< Lectura de una tarjeta del NFC */
    MSG_LCD,   /**< Mensaje de entrada del LCD */
    MSG_WEB,   /**< Mensaje de entrada de la web */
    MSG_RTC,   /**< Mensaje de entrada del RTC */
    MSG_CONS,  /**< Mensaje de entrada del consumo */
    MSG_RADIO, /**< Mensaje de entrada de la radio */
} msg_ctrl_type_t;

/**
 * @brief Estructura para los mensajes de entrada
 */
typedef struct {
    msg_ctrl_type_t type; /**< Tipo de mensaje de entrada.

```

```

Dependiendo de este valor se debe interpretar el contenido */
union {
    nfc_msg_t nfc_msg; /*< Contenido de un mensaje de tipo MSG_NFC */
    lcd_msg_t lcd_msg; /*< Contenido de un mensaje de tipo MSG_LCD */
    rtc_msg_t rtc_msg; /*< Contenido de un mensaje de tipo MSG_RTC */
    web_msg_t web_msg; /*< Contenido de un mensaje de tipo MSG_WEB */
    uint16_t cons_msg; /*< Contenido de un mensaje de tipo MSG_CONS */
    uint32_t radio_msg; /*< Contenido de un mensaje de tipo MSG_RADIO */
};
} msg_ctrl_t;

```

Como se puede ver, dependiendo del valor del campo `type`, tenemos un contenido u otro. Esto permite reducir significativamente la estructura pero requiere de una comprobación de qué tipo de mensaje es antes de acceder a los campos, ya que si no se podrían malinterpretar los mensajes. Se puede ver el contenido de cada tipo de mensaje en el código del Apéndice D.

Este módulo actúa como una inteligencia central de redirección de eventos. A excepción de los GPIO de selección de canal, el módulo únicamente redirige eventos de un módulo a otros, realizando una conversión de datos en los casos que es necesaria.

Por ejemplo, cuando alguien cambia una banda del ecualizador en la pantalla táctil, este módulo recibe un mensaje de tipo `MSG_LCD`, por lo que interpreta su contenido como un `lcd_msg`, que a su vez contiene un campo de subtipo de mensaje y una carga útil. En este ejemplo, el subtipo sería `LCD_BANDS` y la carga útil, de dos bytes, contendría la banda seleccionada y la amplitud que se ha cambiado. Entonces, el módulo principal interpretaría este mensaje y realizaría las dos acciones pertinentes: Enviar un mensaje de tipo `WEB_OUT_BANDS` a la web para que se actualice la información y otro al módulo de procesado digital para que reconfiguren los filtros. Una vez finalizado, se volvería a esperar al siguiente mensaje de la cola.

En tiempo de arranque, este módulo se encarga de habilitar el circuito de audio, enciende la radio y el MP3. Además, utiliza el módulo de la SD para leer la configuración almacenada en los ficheros y configurar tanto la lista de canciones como los filtros.

Además, este módulo es el encargado de poner el sistema en modo bajo consumo. Al entrar en modo bajo consumo, bien por la web o por la pantalla táctil, se realiza el siguiente procedimiento:

1. Apagar la radio y el MP3.
2. Deshabilitar el circuito de alimentación.
3. Limpiar las flags del procesador de *Wakeups*.
4. Apagar la pantalla (ver Párrafo 3.3.1.4).
5. Desactivar el *Wakeups* a través del RTC y sus alarmas.
6. Entrar en modo Standby.

Hemos decidido utilizar el modo standby ya que nuestro sistema no necesita mantener ninguna información en memoria dinámica durante la ejecución del sistema. En este modo, se desactivan los periféricos y el procesador, quedando únicamente a la espera de una señal de *Wakeups* para volver a despertarse. Como dicha señal hemos decidido utilizar el botón azul de la placa, ya que está conectado a un puerto con dicha capacidad.

Como el sistema se desactiva por completo y se pierde el contenido de la memoria, el sistema vuelve a partir desde cero, como si se acabara de conectar la alimentación. Esto es un comportamiento favorable, ya que se vuelven a configurar todos los pines, periféricos y direcciones de memoria adecuadamente y permite un menor consumo que los modos *Sleep* o *Stop*.

Otro factor destacable es que toda la configuración de este módulo en cuanto a pines y niveles lógicos, así como la configuración del módulo de medición de consumo se puede ajustar en el fichero `controlConfig.h`, permitiendo una reconfiguración rápida de ambos módulos.

3.3.3. Módulo de control de consumo

El módulo de medición de consumo se encarga simplemente de realizar medidas periódicas y enviarlas al módulo de control.

Dicho módulo utiliza el ADC3 de la placa para medir el consumo que reporta el módulo analógico de consumo (Subapartado 2.2.3). Dicho módulo reporta la corriente con una ganancia de $1,1V/A$, que junto a la resolución del ADC de 12 bits para $3,3 V$, ofrece una resolución efectiva de:

$$S_{cons} = \frac{S_{ADC}}{S_{med}} = \frac{\frac{3,3 V}{2^{12}}}{1,1 V/A} = 244,14 \mu A$$

Para nuestra aplicación, es una precisión más que suficiente.

El hilo arranca la medida del ADC, realiza una espera para dejar tiempo a que se complete, lee el contenido y lo envía a la cola del sistema operativo, volviendo al principio del proceso.

3.3.4. Módulo de acceso concurrente al I2C

La placa STM32F769NI-DISCO ofrece únicamente un bus I2C en sus pines de extensión, por lo que nos vemos obligados a utilizarlo para tanto la radio como el NFC. Por tanto, hemos creado un módulo de protección para el acceso concurrente a este periférico.

Dicho módulo únicamente utiliza un candado de exclusión mutua (**Mutex**) para evitar que ambos módulos intenten acceder a dicho periférico. Hemos creado una interfaz homónima con CMSIS Driver para que no sea necesario modificar el código existente en demasiada medida.

La principal dificultad del diseño de este módulo es la necesidad de cambiar el hilo al que se avisa en la *callback* del periférico I2C, que se llama desde la interrupción cuando ha finalizado la transferencia. Para solucionarlo, hemos tenido que utilizar una variable estática que contiene el identificador del hilo que está actualmente utilizando el periférico, obtenido mediante la función `osThreadGetId` que ofrece el sistema operativo.

Por tanto, el proceso de las funciones de envío y recepción de información es:

1. Cerrar el candado (`osMutexAcquire`).
2. Colocar el ID del hilo actual en la variable estática.
3. Llamar a la función homónima de CMSIS Driver.
4. Esperar a la flag `FLAGS_I2C_DONE` que se envía cuando finaliza la transferencia.
5. Liberar el candado (`osMutexRelease`).

Con este proceso conseguimos evitar una colisión en el acceso al periférico y las posibles condiciones de carrera que conllevaría. Al ser comunicación I2C, no hay problema de que se alternen la comunicación entre los dos módulos ya que los esclavos ignoran los paquetes que no son para ellos.

3.3.5. Módulo de procesado digital

El módulo de procesado digital es el encargado de tomar las muestras, procesarlas y generar la señal de salida. Se utiliza un ADC de la placa para tomar muestras a $48 kHz$ y un DAC para la generación de la salida, a la misma velocidad.

3.3.5.1 Temporización

La temporización de este módulo es crítica, ya que es la que garantiza la calidad de la señal procesada.

Se utiliza un *Timer* de la placa, concretamente el TIM2 para la generación de dicha señal. Para ello, el *Timer* cuenta con una señal específica de sincronización para disparar eventos de conversión, TRGO2.

Para habilitarla, se necesita además configurar el *timer* como *master*, como se puede ver en el siguiente fragmento. Se utiliza un valor de *Prescaler* de 9 y un *Period* de 224, que junto con la velocidad de reloj del sistema de 216 MHz y el divisor de APB2 de 2, provocan una velocidad final de:

$$f_s = \frac{f_{CLK}}{(PRESCALER + 1)(PERIOD + 1) \cdot Div_{APB2}} = \frac{216 \cdot 10^6}{(9 + 1)(224 + 1) \cdot 2} = 48 \text{ kHz}$$

Fragmento 2: Configuración del TRGO del *timer* de sincronización

```
TIM_ClockConfigTypeDef sClockConfig = { .ClockSource = TIM_CLOCKSOURCE_INTERNAL };
if (HAL_TIM_ConfigClockSource(&htim, &sClockConfig)) {
    return -1;
}

TIM_MasterConfigTypeDef sMasterConfig = {
    .MasterOutputTrigger = TIM_TRGO_UPDATE,
    .MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE,
};
if (HAL_TIMEx_MasterConfigSynchronization(&htim, &sMasterConfig)) {
    return -1;
}
```

Una vez configurado este timer, habrá que configurar al ADC y al DAC para que realicen las conversiones disparados por dicha señal, como se verá en los siguientes apartados.

3.3.5.2 Adquisición de audio

La adquisición de audio, como ya se ha comentado, se realiza a través de un ADC. Los parámetros más importantes de la configuración son:

1. Adquisición a través del GPIO A6.
2. Uso del canal 6 de muestreo.
3. Resolución de 12 bits.
4. Trigger con el TRGO del TIM2.
5. Alineación de datos a la derecha.
6. Peticiones continuas a DMA habilitadas.

Además, se configura el canal 0 del *Stream* 4 de la DMA2 para este ADC. Esto significa que se utilizará este periférico para mover las muestras constantemente del ADC a la memoria sin necesitar ciclos del microcontrolador, lo cual alivia seriamente la carga que recae sobre el procesador. Esta DMA se arranca indicándole un *buffer* de memoria e informa mediante una interrupción de cuando este está a la mitad de capacidad y cuando está completamente lleno. Se configura con los siguientes parámetros:

1. Dirección de periférico a memoria.
2. Incremento de dirección de periférico deshabilitado.
3. Incremento de dirección de memoria habilitada.
4. Alineación de media palabra (16 bits).
5. Modo circular.
6. FIFO desactivada.

Se configura incrementando solo la dirección de memoria ya que el ADC solo tiene una salida (por tanto no tiene sentido incrementarlo) pero interesa que se llene la zona de memoria en lugar de sustituir constantemente el mismo dato. Además, el modo circular provoca que al llegar al final del buffer se vuelva a empezar, permitiendo un flujo constante de información. Esto es muy útil como se verá en los siguientes apartados.

3.3.5.3 Generación de señal

La señal la generamos a través de un **DAC** de la placa. A pesar de que no se indica en la documentación de la placa, en el *datasheet* del procesador vemos que el pin PA4 está conectado al canal 1 del **DAC**, por lo que lo utilizamos para generar la señal de salida.

La configuración más significativa de dicho periférico es:

1. Uso de **GPIO PA4**.
2. Trigger a partir del **TRGO** del **TIM2**.
3. Canal 1 de salida.

Se utiliza también una **DMA** para que el **DAC** tome muestras de una zona de memoria sin necesitar que el procesador se las configure. La configuración más relevante de dicho periférico es:

1. Dirección de memoria a periférico.
2. Incremento de dirección de periférico deshabilitado.
3. Incremento de dirección de memoria habilitada.
4. Alineación de media palabra (16 bits).
5. Modo circular.
6. FIFO desactivada.

Esta **DMA** es de dirección de periférico a memoria, pero el resto de configuración es igual y por el mismo motivo que la del **ADC**.

3.3.5.4 Sistema de Doble Buffer

Para realizar el procesamiento de audio, la implementación más directa es utilizar un *buffer* de memoria y, cuando se llene gracias al **ADC**, realizar un procesamiento de su contenido y sacarlo a través del **DAC**. Sin embargo, esto provocaría que hubiera cortes constantes mientras se procesa el audio.

Para solucionarlo, la solución más común es el uso de un mecanismo de doble *buffer* o mecanismo de *ping-pong*. En este algoritmo, se tienen dos *buffers* de memoria (o uno con el doble de tamaño como en nuestro caso). Mientras se llena uno de los dos, se toman las muestras de el otro y se procesan. Cuando se llena por completo, se comienza a llenar el otro y se procesan las muestras del primero. [15]

Para nuestro caso, necesitamos realizar esta construcción dos veces, una para el **ADC** y otra para el **DAC**. Por tanto, mientras una mitad se va llenando de muestras en un lado y reproduciendo en el otro, la otra mitad del *buffer* del **ADC** se procesa y se coloca el resultado en la otra mitad del **DAC**.

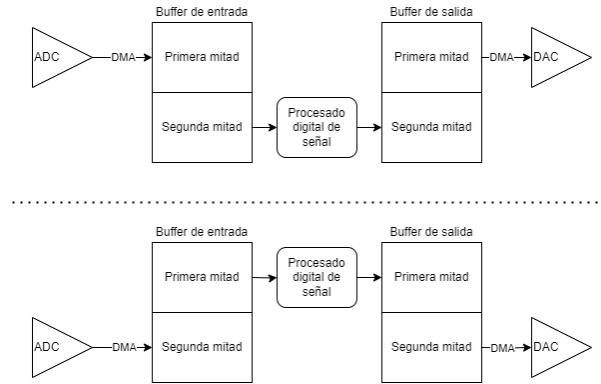


Figura 32: Mecanismo de doble *buffer* para audio

Este mecanismo nos permite reducir la carga del procesador sin perder calidad ni tener cortes notables en el audio.

Para implementarlo, se tiene un hilo de procesado de datos que está constantemente esperando a un par de *flags* que se activan cuando se produce alguna de las interrupciones del DMA del ADC. Cuando estas se producen, se comprueba cual ha sido y se procesa la zona de memoria que corresponda.

3.3.5.5 Sistema de representación de coma fija

Para el procesado de señal hemos utilizado la librería CMSIS DSP, la cual abstrae el procesamiento de señales y las matemáticas con muchos números en los procesadores de arquitectura ARM. [16]

Para optimizarlo, permite la utilización de instrucciones SIMD las cuales realizan el mismo cálculo con muchos datos en un solo ciclo de reloj. Esto permite agilizar significativamente el procesado de grandes zonas de memoria, como pueden ser señales o vectores.

El principal problema de la librería son los formatos de representación con los que trabaja. No permite la utilización de números enteros normales, sino que obliga a utilizar números en coma flotante (más lentos y menos precisos) o en coma fija.

Para los números en coma fija, ARM utiliza el formato de representación de números Q. En este formato, los números se denotan como QX.Y, donde X es el número de bits que representan la parte entera y Y los bits que son parte decimal. Además, hay que indicar si el número es con signo o sin signo, si fuera con signo habría que añadir un bit que depende de la notación se debería incluir en la X o no (en el caso de ARM, la X sí incluye el bit de signo). Para manejar los números negativos se utiliza el complemento a dos.[17]

Concretamente en esta librería, se fija la parte entera a 1 bit (el de signo), por lo que el rango de números representables es aproximadamente $[-1, 1]$. Además, como siempre se utiliza el mismo número, directamente se omite, dando lugar a nombres como Q31, un número de 32 bits en los que el primero es el signo y los demás son parte decimal.

Resumiendo, en esta librería un número x donde b_n es el n-simo bit de menor peso, representado en QN es realmente:

$$x = -b_N + \sum_{i=0}^{N-1} b_i \cdot 2^{i-N}$$

Se puede ver que este formato de representación es equivalente a dividir el formato de representación binario entre 2 elevado a el número de bits menos uno, que coincide con la N del formato:

$$x_{QN} = \frac{x_{bin}}{2^N} = \frac{-2^N \cdot b_N + \sum_{i=0}^{N-1} b_i \cdot 2^i}{2^N} = -b_N + \sum_{i=0}^{N-1} b_i \cdot 2^{i-N}$$

Por tanto, se tiene un isomorfismo entre los dos formatos de representación que además es una operación lineal. Por tanto, todas las operaciones lineales se comportan bien con dicho isomorfismo, es decir, si se pasa un número de un formato a otro, se realiza una operación y se devuelve, es equivalente a hacer la operación en el formato original. Esto es así para las operaciones lineales que nos interesan, es decir, suma y multiplicación por una constante.

Como esas dos operaciones funcionan correctamente, se pueden realizar las siguientes operaciones utilizando números enteros de N bits como si fueran QN:

1. Suma de números.
2. Producto de números.
3. Desplazamiento de números mientras no haya overflow (realmente equivalente a producto por potencia de dos).
4. Convolución, al ser básicamente una suma ponderada.
5. Aplicación de filtro lineal, al poder reducirse a una convolución por la respuesta al impulso del filtro.

Por tanto, y como conclusión, se pueden utilizar las muestras de 12 bits del ADC que están almacenadas como números enteros de 16 bits como números Q15 sin problema, por lo que la librería es ideal para nuestra aplicación.

Para realizar pruebas y comprobar las propiedades de este formato de representación hemos utilizado una calculadora online que permite trabajar con números enteros, en coma fija y sus representaciones binarias.²

3.3.5.6 Procesado Digital de la Señal

Para el procesado de la señal, necesitamos un sistema que aplique ecualización de cinco bandas y permita ajustar el volumen del sistema.

La ecualización se realiza mediante filtros biquadrados, los cuales se caracterizan por configurarse con seis coeficientes: $(a_0, a_1, a_2, b_0, b_1, b_2)$. Con estos seis coeficientes se construye su función de transferencia discreta en el dominio de la transformada z:

$$H(z) = \frac{b_0 + b_1 \cdot z^{-1} + b_2 \cdot z^{-2}}{a_0 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2}}$$

Mediante filtros de esta forma se construyen filtros de tipo *Peak EQ* o *Shelf EQ*, los cuales incrementan o decrementan una banda de frecuencias dejando el resto sin alterar. La utilización de cinco de estos filtros en cascada permite la construcción de un ecualizador de cinco bandas. Para el cálculo de los coeficientes, hemos utilizado los que ofrece la propia librería en su documentación de referencia, específicamente para una aplicación equivalente a esta. [18]

Los coeficientes que ofrecen están en formato Q31, por lo que necesitamos utilizar filtros de dicha longitud. Esto no es un problema e incluso mejora la precisión intermedia de los cálculos.

En CMSIS DSP, se utilizan filtros del tipo `arm_biquad_cas_df1_32x64_q31` para los primeros dos filtros para tener más precisión en las bajas frecuencias y otros de tipo `arm_biquad_cascade_df1_q31` para el resto. Esto implica que cada banda cuenta realmente con dos etapas de filtros en cascada. Además, se fija el valor del coeficiente a_0 a 1 para simplificar las cuentas.

Por tanto, cada etapa necesita diez coeficientes en formato q31 para su inicialización. Además, cada etapa necesita almacenar las variables de estado que se generan en cada iteración y que sirven para calcular la siguiente, por lo que se necesitan dos arrays de 8 elementos de tipo q63 y tres arrays de 8 elementos de tipo q31 adicionales.

Además, se necesitarán convertir las muestras de números q15 a q31 para realizar los cálculos, por lo que se tiene que crear un buffer intermedio del mismo tamaño que el de elementos a procesar.

La gestión de coeficientes es estática, es decir, no se puede cambiar un filtro una vez inicializado. Por ello, cuando se quiere cambiar la amplitud de una banda se necesita reinicializar el filtro.

En total, se tienen diez coeficientes por banda y ganancia, cinco bandas y diecinueve valores posibles de ganancia por banda (desde -9 hasta 9 dB), por lo que se requieren 950 coeficientes distintos.

Además, como el formato de representación permite únicamente valores de módulo inferior a la unidad, se utiliza un *shift* que multiplica todos los coeficientes por una potencia de dos.

Para el volumen se utiliza la función `arm_scale_q31`, que permite multiplicar toda una zona de memoria por un número, siempre que no sea máximo, que no se multiplica el número por nada. Como hemos explicado en el apartado anterior, para que la multiplicación en el dominio binario natural se mantenga, hay que multiplicar por la equivalencia en Q31 de los números del 0 al 1 en pasos de 0,1. Como adición, si el volumen es cero, se utiliza la función `arm_fill_q15` para llenar el buffer con el número intermedio del rango (2048) y así reducir computaciones innecesarias que implican un consumo de potencia inútil.

Además, antes de procesar la señal se utiliza la función `arm_offset_q15` para restar el offset y añadirlo después de los cálculos y se utiliza `arm_shift_q31` para desplazar los datos y permitir así tener más margen de precisión en los cálculos.

²<https://chummerson.github.io/qformat.html>

Finalmente, se eliminan los valores superiores a 4095 e inferiores a 0 ya que como solo se utilizan 12 de los 16 bits, estos valores provocan un overflow que cambia bruscamente la salida del DAC.

Se recoge la función de procesado en el siguiente fragmento de código.

Fragmento 3: Función de procesado de audio

```

void processSamples(uint16_t* in, uint16_t* out) {
    if (volume == 0) { // Volumen 0 -> No procesar salida (menos consumo)
        arm_fill_q15(2048, (q15_t*)out, DSP_BUFSIZE);
        return;
    }

    arm_offset_q15((q15_t*)in, -2048, (q15_t*)out, DSP_BUFSIZE);

    arm_q15_to_q31((q15_t*)out, midBuffer, DSP_BUFSIZE);

    arm_shift_q31(midBuffer, -DSP_SHIFT_MARGIN, midBuffer, DSP_BUFSIZE);

    if (volume < 10) { // Aplicar volumen
        arm_scale_q31(midBuffer, volCoeffs[volume], 0, midBuffer, DSP_BUFSIZE);
    }

    // Aplicar filtros
    for (int i = 0; i < 2; i++) {
        arm_biquad_cas_df1_32x64_q31(&lowStageHandlers[i], midBuffer, midBuffer, DSP_BUFSIZE);
    }
    for (int i = 0; i < 3; i++) {
        arm_biquad_cascade_df1_q31(&highStageHandlers[i], midBuffer, midBuffer, DSP_BUFSIZE);
    }

    arm_shift_q31(midBuffer, DSP_SHIFT_MARGIN, midBuffer, DSP_BUFSIZE);

    arm_q31_to_q15(midBuffer, (q15_t*)out, DSP_BUFSIZE);

    // Recuperar offset
    arm_offset_q15((q15_t*)out, 2048, (q15_t*)out, DSP_BUFSIZE);

    // Saturar valores fuera de rango
    arm_clip_q15((q15_t*)out, (q15_t*)out, 0, 4095, DSP_BUFSIZE);
}

```

3.3.5.7 Fichero de configuración

Al igual que el módulo de control, la configuración de este módulo se puede ajustar mediante un fichero de cabecera, llamado `audioConfig.h`. En él, se pueden ajustar los pines utilizados en cada periférico, los canales utilizados y la temporización mediante los controles del timer.

Este fichero nos ha permitido utilizar el mismo código para realizar una primera implementación en la placa que cuenta con un procesador Cortex M4.

3.3.6. Módulo NFC

Permite que se seleccionen canciones y emisoras mediante el móvil con NFC.

Los ficheros más destacables del módulo son:

1. `nfc.c`: Hilo que gestiona el uso de la tag. Los datos obtenidos se envían por una cola al hilo de control.
2. `nfc.h`: Exporta la función para el arranque del hilo.

El principal problema a la hora de utilizar este componente, es que no pueden estar activos al mismo tiempo, por lo que hemos tenido que adaptar la utilización de estos dos protocolos con el uso real de nuestro módulo NFC. A la hora de guardar y leer datos, en ambos casos, se guardan en NDEF.

Nosotros vamos a utilizar RF mediante el uso del teléfono móvil para escribir en el NDEF, mientras que el I2C para leer el contenido, por lo que solo contaremos con esos uso. Para escribir mediante RF, la transferencia consiste en dos pulsos a nivel bajo en el GPO.

Por esto mismo, el código consiste en habilitar el RF (DIS = 0), esperar a una subida en el GPO y esperar 500 ms (Asegurando que se ha completado la transferencia). Después, se deshabilita el RF (DIS = 1) y se procede a enviar todas las transmisiones por I2C. Finalmente, se envía la información al hilo de control y se procede a habilitar de nuevo el RF y volver a esperar al flanco de subida.

A la hora de trabajar con el I2C, utilizaremos la familia de comandos NFC Forum Type 4 Tag, mientras que con RF se utiliza ISO/IEC 7816-4. En lo referente a I2C, tenemos que crear las tramas correspondientes a cada comando. Nosotros utilizaremos:

1. NDEF Tag Application Select: Selecciona la aplicación NDEF Tag.
2. NDEF Select: Selecciona el fichero NDEF.
3. Read Binary: Lee datos de un fichero.

Los utilizaremos de la siguiente manera:

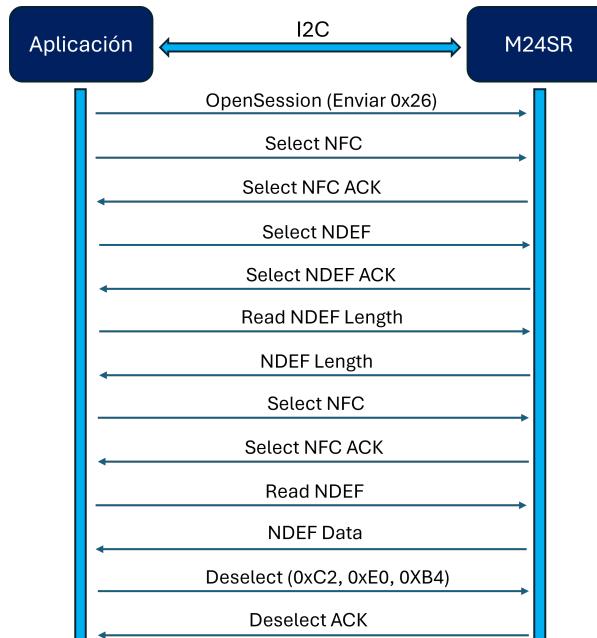


Figura 33: Diagrama de flujo de una operación de lectura con el NFC

Todos los bytes para las transferencias los hemos calculado gracias al datasheet del componente M24SR64-Y. Además, antes de enviar el master recieve, hay que esperar por los tiempos de la tarjeta.

El módulo también utiliza CRC para la detección de errores en transferencias I2C. Cada byte I2C se calcula utilizando ese algoritmo, y al final de la trama se envían 2 bytes con el valor calculado. Para calcularlo, hemos utilizado la siguiente página web [19], introduciendo todos los bytes de la trama.

Los mensajes que hemos definido para el NFC están compuestos por una letra (S = canción, R = radio) y 4 números (Nº canción o emisora de radio).

Ejemplos:

- S 0014: Se quiere escuchar la canción nº 14.
- R 1029: Se quiere escuchar la emisora 102.9 FM.

3.3.7. Módulo de la SD

Se encarga al inicio de la aplicación, de acceder a la tarjeta uSD para obtener la lista de canciones y la configuración guardada. Cuando se le indica, se encarga de guardar la configuración actual para poder iniciarse en el próximo encendido con la misma configuración.

Para su desarrollo, primero intentamos utilizar el driver SD de la placa STM32F769I-Disco, la librería *FatFs* [20][21] y una tarjeta micro-SD.

Hemos decidido almacenar en la tarjeta el siguiente contenido:

- **songs.txt**: Lista de canciones. Máximo puede haber 25 canciones, de 30 caracteres cada una (29 en Windows debido al salto de línea).
- **config.txt**: Configuración del sistema. Valor de las 5 bandas de ecualización y del volumen.

La estructura del código necesario para este proyecto es:

- FatFs:

- ff.c, ff.h
- ff_gen_drv.c, ff_gen_drv.h
- diskio.c, diskio.h
- sd_diskio.c, sd_diskio.h

- uSD:

- sd.c, sd.h
- fatfs_storage.c, fatfs_storage.h

FatFS es un módulo genérico de sistema de archivos FAT/exFAT para pequeños sistemas embebidos. Es independiente de la plataforma en la que se utilice. Está compuesto por las librerías mencionadas en la estructura.

FatFs se utiliza como abstracción sobre el sistema de ficheros Fat. En nuestro nivel de aplicación solo necesitamos llamar a las funciones de fichero **ff**. Sin embargo, tenemos que crear una clase que permita unir las librerías FatFs (mediante **diskio.h**) con la implementación hardware de la uSD. Nosotros utilizamos para ello la clase **sd_diskio**, mediante el Board Support Package (BSP) de la STM32F769I-Disco.

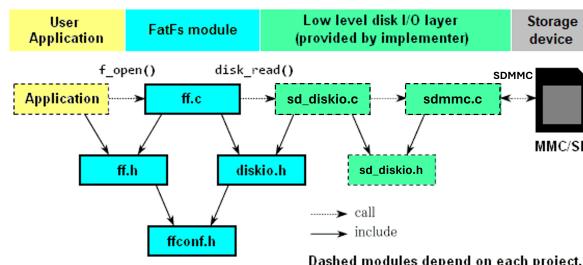


Figura 34: Configuración del sistema implementando FatFs

Los ficheros **sd.c** y **sd.h** se encargan de inicializar la tarjeta, obtener la lista de canciones y la configuración guardada, y de guardar la configuración actual. Esto lo hace utilizando las funciones definidas en **fatfs_storage.h**. Además, se encarga de revisar que los valores son correctos y están en el rango adecuado. Si no están dentro del rango, los modifica al valor más cercano dentro del rango. También se encarga de adaptar todo lo obtenido a los tipos de variables correctos para la integración del proyecto.

En primer lugar, se inicializa la uSD con la función pública **Init_SD**. Basicamente se monta la tarjeta y se obtienen el directorio, las canciones (**Get_Songs**) y la configuración guardada (**Get_Config**). Toda la información se guarda en punteros a variables del control. Después, cuando se quiere guardar la configuración, se llama a la función pública **Save_Config**.

Para parsear toda la información lo hacemos de la siguiente manera:

- Parseo de la lectura de la configuración: Utilizamos un `sscanf`.
- Parseo de la lectura de las canciones: Utilizamos un algoritmo de copia de cadena de caracteres de triple puntero.
- Parseo de la escritura de la configuración: Utilizamos un `sprintf`.

`fatfs_storage.c` y `fatfs_storage.h` se encargan de las operaciones de lectura y escritura de los archivos. Esto lo lleva a cabo mediante llamadas a las funciones de las librerías de FatFs. Tiene tres funciones diferentes:

- `Storage_GetDirectoryBitmapFiles`: Se monta la tarjeta con `f_mount` y se obtiene el directorio de ficheros. El directorio se obtiene llamando a la función de FatFs `find_first` y después con `find_next` mientras existan más ficheros. Al final, se cierra el directorio con `closedir`. (`f_mount → f_findfirst → f_findnext → f_closedir`).
- `Storage_OpenReadFile`: Se lee un archivo. Primero se abre el archivo, después se lee y finalmente se cierra. (`f_open → f_read → f_close`).
- `Storage_OpenWriteFile`: Se escribe en un archivo. Primero se abre el archivo, después se desplaza el puntero de escritura al principio del fichero .txt y se escribe en él. Finalmente se cierra. (`f_open → f_lseek → f_write → f_close`).

Sin embargo, a la hora de realizar la integración con el resto del proyecto, no funcionaba. La tarjeta llegaba a montarse de manera correcta, pero a la hora de buscar los ficheros, devolvía un `RXOVER`, que indica que la transferencia de lectura ha sufrido de overflow.

Por tanto, hemos intentado

Desde RTAP, se interactúa con la SD a través del periférico SDMMC2, configurándose el *bus width* a 1 solo bit. Esta configuración se elige porque, pese a que el periférico permite la ejecución de operaciones de entrada/salida con hasta 8 bits en paralelo, resulta una complicación innecesaria que puede derivar en un error: La SD puede ser más rápida que el microcontrolador, perdiendo la sincronización y fracasando en las operaciones. Además, como desde RTAP las canciones solo se leen una vez, al inicio del programa, la velocidad con la que se realicen las operaciones de I/O no afectan a la experiencia del usuario.

Para el manejo de este periférico, nos ayudamos del *Software Pack* de Keil, llamado *File System*. Este *Software Pack* ofrece una abstracción de las funciones de más bajo nivel, eliminando la necesidad de interactuar con registros directamente. Se pueden utilizar las funciones presentes en `<stdio.h>` para el manejo de archivos, siendo especialmente relevantes `fopen`, `fread` y `fwrite`.

3.3.8. Módulo RTC

El principal objetivo de este módulo es mostrar la hora y la fecha actual en el sistema. Esto se consigue mediante el periférico RTC del propio microcontrolador.

Dicho reloj se ha configurado para que se conecte al oscilador interno LSE, que cuenta con una frecuencia de 32.768kHz. Debido a esta frecuencia, se ha configurado el periférico con unos valores `AsynchPrediv = 127` y `SynchPrediv = 255` para obtener un periodo de 1 segundo.

Por otra parte, para obtener la hora actual, se ha optado por utilizar una sincronización con un servidor mediante SNTP.

También se ha configurado el RTC para generar una interrupción mediante una alarma con una frecuencia de 1Hz. En el callback de dicha alarma, se informa a este módulo, mediante una flag, que debe indicar al módulo principal, la fecha y la hora actuales. Dicha comunicación se realiza mediante una cola de mensajes.

3.3.9. Módulo Radio

El módulo del Sintonizador FM consiste en un *Thread* que se encarga de gestionar el funcionamiento del propio módulo. La comunicación entre el periférico y el microcontrolador se ha configurado mediante el *Driver I2C* proporcionado por CMSIS.

Cuenta con dos colas de mensajes, una en la que el programa principal introduce los mensajes con los comandos que desea que ejecute el Sintonizador, como sintonizar una frecuencia, hacer un *Seekup* o un *SeekDown*, etc. La otra cola se utiliza para que la radio introduzca mensajes con información sobre las frecuencias sintonizadas. Los mensajes mandados por esta cola son del tipo *MSG_RADIO*.

También cuenta con un *timer* periódico, con un periodo de medio segundo, que se encarga de leer los registros del sintonizador para comprobar su correcto funcionamiento.

Debido a que nuestro sistema cuenta con dos periféricos que utilizan el protocolo I₂C, nos hemos visto obligados a utilizar un módulo de gestión del propio protocolo de comunicación, del cual se hablará en el Subapartado 3.3.4.

El comportamiento básico de este módulo consiste en una espera mediante un *osMessageQueueGet* de un mensaje proporcionado por el programa principal. Una vez el mensaje es recibido, es procesado y en función de su contenido, se ejecutará el comando correspondiente. En caso de ser un *SeekUp* o un *SeekDown*, el módulo del sintonizador informará al *Thread* principal para que muestre la frecuencia en la que ha terminado.

3.3.10. Módulo MP3

El módulo del reproductor MP3 consiste en un *Thread* que se encarga de controlar la gestión del propio módulo.

Cuenta con una cola de mensajes en la que el programa principal introduce mensajes con el comando a ejecutar. Algunos de estos comandos son reproducir una canción, pausar su reproducción, siguiente canción, etc.

Para gestionar la comunicación entre este módulo y el principal, se ha utilizado la USART6 mediante el *Driver_USART6* proporcionado por CMSIS. Dicha USART se ha configurado atendiendo a las necesidades del reproductor, modo asíncrono, datos de 8 bits sin paridad, utilizando 1 bit de stop y con una velocidad de 9600 bps.

Este módulo no envía mensajes al programa principal debido a que el reproductor no ofrece la posibilidad de leer sus registros mediante la UART.

El comportamiento básico de este módulo consiste en una espera mediante un *osMessageQueueGet* de un mensaje proporcionado por el programa principal. Una vez el mensaje es recibido, es procesado y en función de su contenido, se ejecutará el comando correspondiente. Una vez la canción es seleccionada, se realiza otra espera mediante un *osThreadFlagsWait* hasta que la transferencia es completada, en cuyo caso el módulo continua con su funcionamiento.

3.3.11. Módulo Web

Para la generación de las diferentes páginas web del servidor y la creación de los diferentes scripts para interactuar con dichas páginas, hemos creados varios archivos con extensión CGI como el que se adjunta a continuación:

```
t      <form action="index.cgi" method="post">
c i 1      <input type="radio" id="entrada_radio" name="entrada" value="radio"
            OnClick="submit();" %s>
t          <label for="entrada_radio" style="font-size: 20px;">Radio</label>
c i 2      <input type="radio" id="entrada_mp3" name="entrada" value="mp3"
            OnClick="submit();" %s>
t          <label for="entrada_mp3" style="font-size: 20px;">MP3</label>
t      </form>
t      <br><br>
t      <form action="index.cgi" method="post">
c i 3      <input type="radio" id="salida_altavoz" name="salida" value="altavoz"
            OnClick="submit();" %s>
t          <label for="salida_altavoz" style="font-size: 20px;">Altavoz</label>
c i 4      <input type="radio" id="salida_cascos" name="salida" value="cascos"
            OnClick="submit();" %s>
```

```
t      <label for="salida_cascos" style="font-size: 20px;">Cascos</label>
t    </form>
```

Fragmento 4: Ejemplo archivo .CGI

Como se puede observar, existen dos tipos diferentes de líneas de código, las que empiezan por “t” y las que empiezan por “c”. Las que empiezan por “t” son ignoradas por el compilador y no se procesan, en cambio, las que empiezan en por “c” son atendidas y procesadas. A continuación, se comprueba la letra siguiente al espacio, en este caso la “i”, debido a que nos encontramos en el archivo *index.cgi*.

Por último, se obtiene el siguiente carácter a continuación del espacio en blanco y, el propio programa del servidor web, tomará unas medidas u otras dependiendo de dicha letra o número.

En todos los casos, se sustituirá el “%s” que encontramos al final de dichas líneas de código por el conjunto de caracteres deseado mediante las función *sprintf()*. A continuación se adjunta un trozo de código del programa principal del servidor web en el que se realiza dicha acción:

```
switch(env[0]){
    case 'i':
        // Cases for index
        switch (env[2]){
            case '1':
                // Case for Radio Input
                len = sprintf (buf, &env[4], web_state.entrada == WEB_RADIO ? "checked" : "");
                break;
            case '2':
                // Case for MP3 Input
                len = sprintf (buf, &env[4], web_state.entrada == WEB_MP3 ? "checked" : "");
                break;
            case '3':
                // Case for Altavoz Output
                len = sprintf (buf, &env[4], web_state.salida == WEB_ALTAZOZ ? "checked" : "");
                break;
            case '4':
                // Case for Auriculares Output
                len = sprintf (buf, &env[4], web_state.salida == WEB_AURICULARES ? "checked" : "");
                break;
        }
        break;
}
```

Fragmento 5: Ejemplo procesamiento archivo .CGI

Por otra parte, para enviar los datos desde las páginas al programa principal del servidor, se ha optado por usar fórmularios con el método *post*, los cuales se envían si se pulsa algún botón mediante la función *OnClick="submit()"*.

Por último, vamos a comentar la función utilizada para actualizar periódicamente tanto la fecha y la hora como el consumo medido. Esto se ha realizado mediante una función creada en JavaScript llamada *periodicUpdate()*. A continuación se adjunta dicha función en la página denominada *index*:

```
<script language=JavaScript type="text/javascript" src="xml_http.js"></script>
<script>

var timeUpdate = new periodicObj("time.cgi", 500);
function periodicUpdateRTC(){
    updateMultiple(timeUpdate, plotRTCTime);
    rtc_elTime = setTimeout(periodicUpdateRTC, timeUpdate.period);
}
function plotRTCTime(){
    timeVal = document.getElementById("rtcTime").value;
    document.getElementById("rtc").textContent = timeVal;
    consumo = document.getElementById("cons_ref").value;
    jg1.refresh(consumo);
}
```

```
    }
</script>
```

Fragmento 6: Función updatePeriodic()

Como se puede observar, dicha función consiste en una llamada, cada 500ms, al archivo *time.cgi* de donde se obtienen periódicamente los valores de fecha y hora y de consumo. Cuando dichos valores son obtenido, se actualizan sus valores mostrados en las diferentes páginas web. A continuación se adjunta el código presente en el archivo *time.cgi*, el cual tiene un comportamiento idéntico a los archivos con extensión .CGI antes mencionados:

```
t <form>
t <text>
t <id>rtcTime</id>
c h <value>%02d-%02d-20%02d %02d:%02d:%02d</value>
t </text>
t <text>
t <id>cons_ref</id>
c z <value>%04d</value>
t </text>
t </form>
.
```

Fragmento 7: Archivo time.cgi

4. Depuración y test

4.1. Pruebas software

4.1.1. Pruebas del módulo de procesado digital de señal

Para probar el módulo de procesado digital de señal, se conecta un generador de señal a la entrada del ADC de la placa y se mide la salida del DAC mediante un osciloscopio.

Primero probamos si el túnel de audio funciona, para lo cual no habilitamos ningún filtro. Como se ve en la Figura 35, se tiene exactamente la misma señal pero retrasada unos milisegundos.

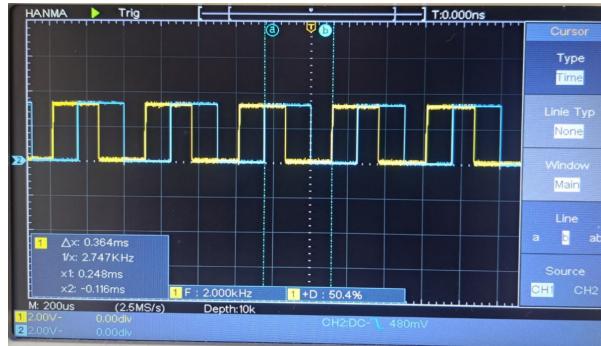


Figura 35: Camino de audio sin procesado

Después, probamos por ejemplo a reducir las altas frecuencias y bajar un poco el volumen, obteniendo una salida como la de la Figura 36.

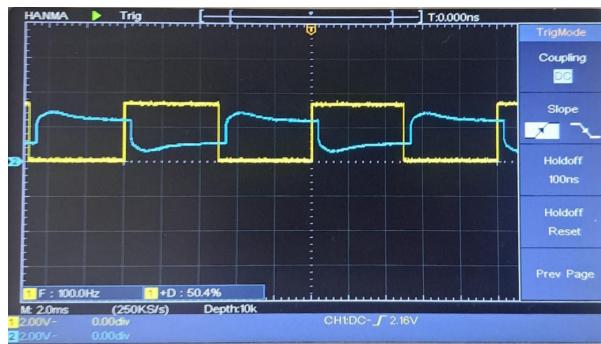


Figura 36: Señal cuadrada procesada digitalmente

4.1.2. Test Radio

El objetivo de esta prueba es la comprobación del correcto funcionamiento del módulo del sintonizador FM. Para navegar entre las direfentes pruebas de este test, se ha utilizado el botón azul (B1) de la propia placa.

Para ello, primero se manda el comando, mediante un *Thread* auxiliar, para encender la radio y se comprueba si se obtiene señal de audio a la salida.

A continuación, se sintoniza una frecuencia concreta y se comprueba, mediante una radio externa, si ambas señales de audio coinciden.

Ahora, se realiza un *SeekUp* y se comprueba si la nueva frecuencia sintonizada es mayor a la anterior y si la calidad del audio aumenta.

De manera análoga, se realiza un *SeekDown* y se comprueba si la frecuencia obtenida es menor a la anterior y también si la calidad de audio aumenta.

A continuación, se intenta sintonizar una frecuencia fuera de rango y se comprueba que solo se obtiene ruido a la salida.

Por último, se manda el comando para apagar la radio y se comprueba que ya no hay señal de audio a la salida.

4.1.3. Test MP3

El objetivo de esta prueba es la comprobación del correcto funcionamiento del módulo del reproductor MP3. Para navegar entre las diferentes pruebas de este test, se ha utilizado el botón azul (B1) de la propia placa.

Para ello, primero se inicia la reproducción de una canción concreta y se comprueba si a la salida se escucha esa canción.

Ahora, se manda el comando, mediante un *Thread* auxiliar, el comando que indica la reproducción de la siguiente canción de la lista y se comprueba si se obtiene a la salida.

A continuación, se indica al reproductor que ponga la anterior canción y se comprueba si se escucha dicha canción.

La siguiente comprobación es la puesta en pausa de la canción reproducida, para ella se manda dicho comando y se comprueba que no se obtiene salida.

De forma análoga, se le indica al reproductor que continue con la reproducción de la canción y se comprueba que se obtiene la canción esperada.

Ahora, se intenta seleccionar una canción que no esté presente en la lista, comprobando que no se obtiene señal a la salida.

Por último, se comprueba el modo *loop*. Para ello se manda el comando indicado y se espera a que termine la canción actual y se comprueba que vuelve a comenzar y, de forma análoga, se indica al reproductor que termine dicho modo y se comprueba, al finalizar la canción actual, que no se obtiene señal a la salida.

4.1.4. Test RTC

El objetivo de esta prueba es la comprobación del correcto funcionamiento del módulo RTC.

Concretamente, se prueba si el reloj pasa el tiempo correctamente, si se pasa de minuto y hora adecuadamente y si la fecha se actualiza correctamente.

Implícitamente se comprueba el correcto funcionamiento de la alarma y, además, si la frecuencia a la que se activa es correcta.

También se comprueba si la sincronización con el servidor SNTP es correcta y si la hora actualizada coincide con la fecha y hora actuales.

Por último, se comprueba si los mensajes que envía al programa principal son del tipo correcto y si su contenido es el esperado.

4.1.5. Test Web

El objetivo de esta prueba es la comprobación del correcto funcionamiento del módulo del servidor web.

Para ello, se va a dividir este test en dos partes. En la primera, se comprobará si las peticiones que se generan desde las diferentes páginas web se crean de manera correcta.

Para conseguir esto, se van pulsando sucesivamente todos los botones de todas las páginas y se comprueba, en el módulo del servidor, si se generan de manera correcta.

También se comprueba si los mensajes que se generan para las diferentes peticiones creadas son correctos y se envían de forma exitosa al programa principal.

A continuación, se procederá a comprobar si los valores mostrados en las páginas web se actualizan de manera correcta. Para ello, se enviará, desde un *Thread* auxiliar, diferentes modificaciones en los datos mostrado y se comprobará si se actualizan de manera correcta.

Por último, se comprobará si tanto la hora y la fecha como el consumo se actualizan en tiempo real, implicitamente comprobando el correcto funcionamiento de las funciones desarrolladas en JavaScript, por lo que se envían durante un cierto periodo de timepo y con una frecuencia de 1Hz, los valores de tiempo, fecha y consumo comprobando que en las diferentes páginas web se muestra de manera correcta.

4.2. Pruebas hardware

4.2.1. Pruebas de la placa de audio

Para probar la placa de audio, la colocamos sola, alimentándola con una fuente de alimentación de laboratorio, colocando un Jumper entre el terminal de ADC y DAC para no tener en cuenta el procesado digital.

Después, se introduce una señal sinusoidal y se mide la amplitud de salida, calculando la respuesta en frecuencia del circuito. Se recoge una gráfica con la respuesta en frecuencia en decibelios relativos al máximo en la Figura 37 (auriculares) y la Figura 38 (altavoces). Cabe destacar que la respuesta dibujada es en decibelios relativos al máximo, por lo que parece que tienen la misma amplitud, pero el altavoz duplica la amplitud de la señal de entrada.

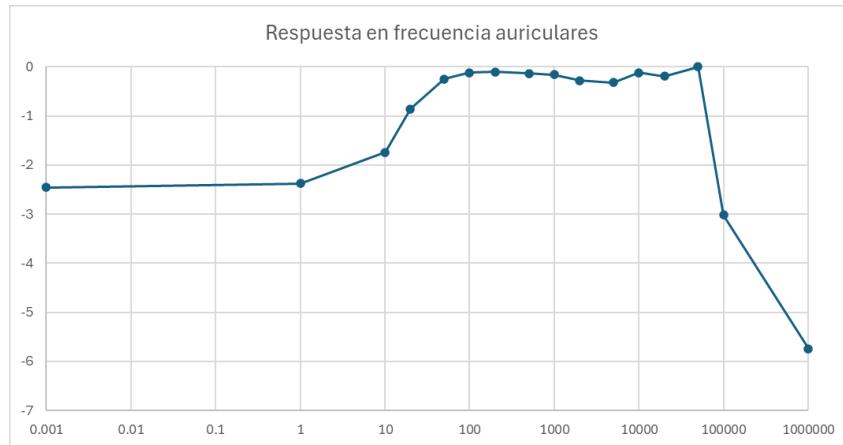


Figura 37: Respuesta del circuito amplificador de auriculares

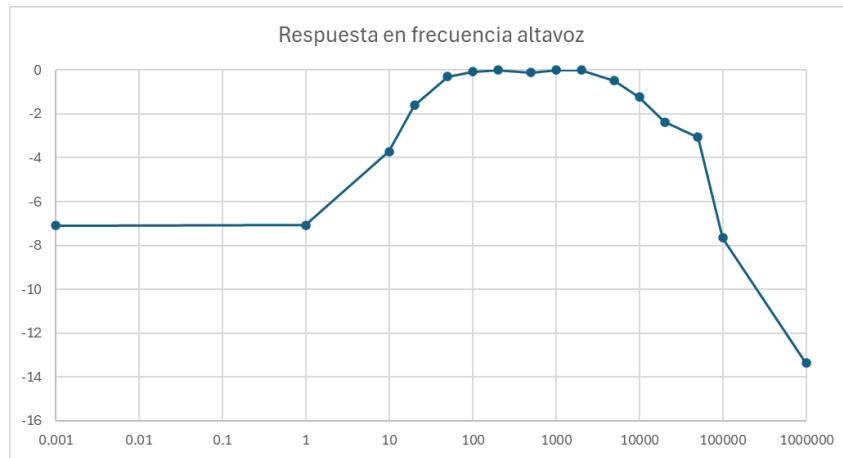


Figura 38: Respuesta del circuito amplificador de altavoces

Comprobando el circuito de habilitación, funciona adecuadamente pero a veces se comporta de forma poco consistente al intentar desactivar el circuito cuando está habilitado, pero generalmente se comporta correctamente.

Por otro lado, conectamos distintas cargas y medimos el consumo del circuito, obteniendo los siguientes valores de consumo en función de la impedancia nominal del altavoz. Obtenemos los siguientes valores:

1. Auriculares de 40Ω : Consumo de 4 mA .
2. Altavoz de 8Ω : Consumo 64 mA .

3. Altavoz de $4\ \Omega$: Consumo de $164\ mA$.

Además, probando el cambiador de nivel, vemos que consigue una tensión de nivel bajo de $19,78\ mV$ y un $6,99\ V$.

4.2.2. Pruebas del circuito de alimentación

Para realizas las pruebas del circuito de alimentación, probamos primero a realizar una descarga y carga de la batería para probar este funcionamiento. Descargamos la batería hasta un valor aproximado de $3,5\ V$ y la volvimos a cargar con nuestro cargador hasta un valor de $4,05\ V$, en el cual la corriente de carga comienza a disminuir y el proceso de carga se ralentiza.

En condiciones normales, la batería carga con una corriente constante de entre 600 y $700\ mA$, pero disminuye en el tramo final como se comentó en el diseño del circuito. También se ha probado a cargar el circuito a la vez que se carga la batería. Si la fuente tiene suficiente capacidad como para alimentar las dos cosas, hemos comprobado que así lo hace. Hemos probado por ejemplo con una carga de $10\ \Omega$, con lo que el consumo de aproximadamente $700\ mA$ se suma a la carga obteniendo cerca de $1,5\ A$.

Por otro lado, caracterizamos la regulación de carga para comprobar que nuestro circuito mantuviera la tensión de salida incluso cuando es demandado una gran cantidad de corriente. Se ve que obtenemos un valor de $-39,012V/A$, o $0,56\ %/A$, un buen valor contando con que el consumo aproximado será de medio amperio. Se recoge la gráfica de las medidas en la Figura 39.

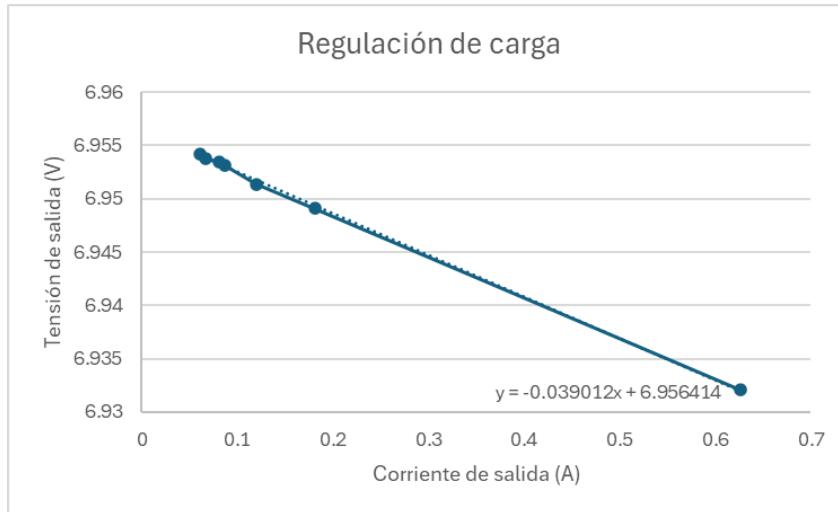


Figura 39: Regulación de carga del circuito de baterías

Además, comprobamos que los indicadores luminosos funcionan correctamente ya que se enciende el verde cuando se conecta la alimentación y el rojo cuando se está cargando la batería. Al finalizar la carga, se apaga la luz roja y si se enchufa sin estar conectada parpadea el indicador rojo.

Mediante el osciloscopio medimos la tensión de salida del circuito de alimentación, observando que presenta un ruido en muy alta frecuencia, como se puede ver en la Figura 40. Creemos que este pico de ruido es el culpable de la inestabilidad de los relojes de la placa, pero al colocar condensadores en paralelo no conseguimos reducirlo.

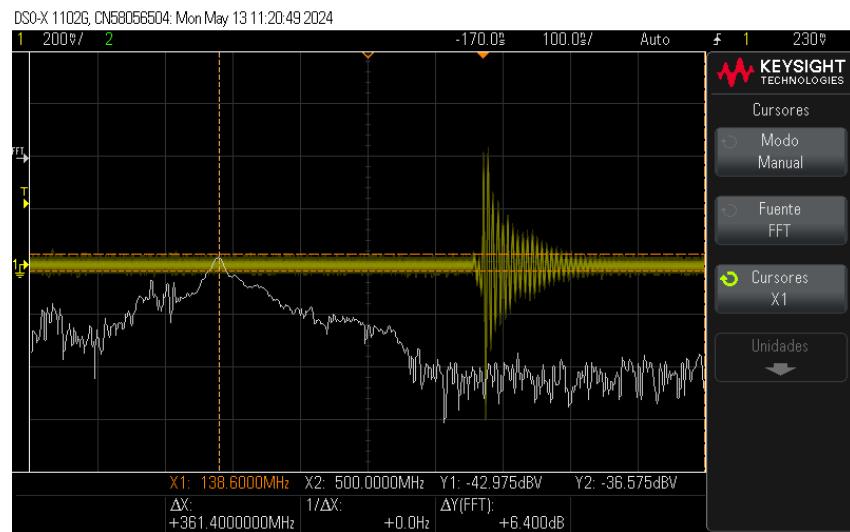


Figura 40: Ruido en la señal de alimentación

5. Presupuesto final

El presupuesto total del proyecto está recogido en la Tabla 1. Como se puede observar, tanto la tarjeta STM32F769NI como el sintonizador FM y el reproductor MP3, han sido cedidos por profesorado de la asignatura, por lo que su precio no se ve reflejado en el presupuesto total.

Concepto	Precio
Sensor NFC	5.00
Componentes Analógicos	65.00
PCBs	40.00
Batería	8.00
Tarjeta STM32F769NI	Cedido
Sintonizador FM	Cedido
Reproductor MP3	Cedido
Total	118.00

Tabla 1: Presupuesto del proyecto

6. Equipo de trabajo

Todos los integrantes hemos colaborado en el diseño de los módulos, pero la responsabilidad principal de cada uno se ha repartido como:

- Rubén Agustín González: Interfaz de la pantalla de la placa, bajo consumo y uSD.
- David Andrino Izquierdo: Esquemático de la alimentación, diseño de las PCBs, módulo de control, protector I2C y procesado digital de señal.
- Estela Mora Barba: Esquemático del amplificador de audio, módulo NFC y uSD.
- Fernando Sanz Giménez: Módulo RTC, radio, MP3 y web.

Las siguientes partes han sido relacionadas de forma conjunta por todos:

- Integración del proyecto.
- Realización de la memoria.
- Powerpoints de las presentaciones.

A la hora de colaborar y juntar todas las partes del proyecto, hemos utilizado un repositorio en GitHub³, con varias ramas creadas para cada miembro. Además, hemos estado trabajando tanto presencial como online, de forma tanto individual como colectiva.

Ha habido mucha colaboración entre los miembros del grupo, no solo en lo referente al propio trabajo sino que también de forma emocional.

³<https://github.com/David-Andrino/ise-rtap>

7. Acrónimos utilizados

Acrónimo	Significado
ADC	Analog to Digital Converter
API	Application Programming Interface
ARM	Advanced RISC Machine
BSP	Board Support Package
CGI	Common Gateway Interface
CMD	Command Prompt
CMSIS	Cortex Microcontroller Software Interface Standard
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DIS	RF disable
DMA	Direct Memory Access
DSI	Display Serial Interface
DSP	Digital Signal Processor
EEPROM	Electrical Erasable Programmable Read-Only Memory
EXFAT	Extended FAT
FAT	File Allocation Table
GFX	Graphics Effects
GND	Ground
GPIO	General Purpose Input/Output
GPO	configurable General Purpose Output
EEPROM	Electrical Erasable Programmable Read-Only Memory
EXFAT	Extended FAT
FAT	File Allocation Table
GPO	configurable General Purpose Output
HAL	Hardware Abstraction Layer
I2C	Inter Integrated Circuit
LVGL	Light and Versatile Embedded Graphics Library
NDEF	NFC Data Exchange Format
NFC	Near Field Communication
NVIC	Nested Vector Interrupt Controller
PWM	Pulse Width Modulation
RCC	Reset and Clock Control
RF	Radio Frequency
RFID	Radio Frequency Identification
RGB	Red Green Blue
RMS	Root Mean Squared
RTAP	Real Time Audio Processor
RTC	Real Time Clock
RTOS	Real Time Operating System
SCL	Serial Clock
SDA	Serial Data
SDMMC	Secure Digital/ MultiMediaCard interface
SDRAM	Synchronous dynamic random access memory
SNTP	Simple Network Time Protocol
TIM	Timer
TTL	Universal synchronous and asynchronous receiver-transmitter
UART	Universal Asynchronous Receiver/Transmitter
USART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
WAV	Waveform Audio File
XFER	Transfer

Tabla 2: Acrónimos del documento

8. Bibliografía

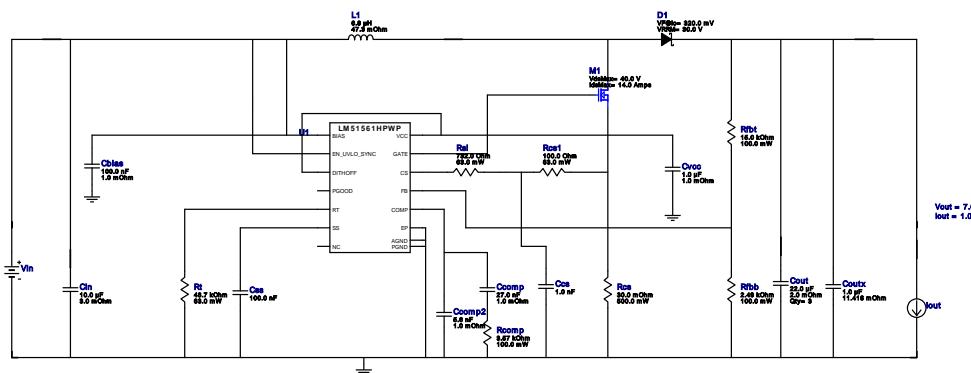
- [1] *BU-409: Charging Lithium-ion*, sep. de 2010. dirección: <https://batteryuniversity.com/article/bu-409-charging-lithium-ion> (visitado 11-05-2024).
- [2] *BQ25606 Data Sheet, Product Information and Support* — TI.Com. dirección: <https://www.ti.com/product/BQ25606> (visitado 11-05-2024).
- [3] Texas Instruments, *Designing A Standalone Single Cell 3-A Charger with the Bq25606*. dirección: <https://www.ti.com/lit/an/slva924/slva924.pdf> (visitado 11-05-2024).
- [4] *LM51561H Data Sheet, Product Information and Support* — TI.Com. dirección: <https://www.ti.com/product/LM51561H?qgpn=lm51561h> (visitado 11-05-2024).
- [5] A. Hari, «Improve Power Converter Reliability Using Hiccup-Mode Current Limiting,» dirección: <https://www.ti.com/lit/an/snva604/snva604.pdf>.
- [6] *OPA187 Data Sheet, Product Information and Support* — TI.Com. dirección: <https://www.ti.com/product/OPA187> (visitado 11-05-2024).
- [7] «Transmission gate,» Wikipedia, jun. de 2023. dirección: https://en.wikipedia.org/w/index.php?title=Transmission_gate (visitado 11-05-2024).
- [8] *CD4053B Data Sheet, Product Information and Support* — TI.Com. dirección: <https://www.ti.com/product/CD4053B> (visitado 11-05-2024).
- [9] *5651_tuner84_RDA5807M_datasheet_v1.Pdf*. dirección: https://cdn-shop.adafruit.com/product-files/5651/5651_tuner84_RDA5807M_datasheet_v1.pdf (visitado 11-05-2024).
- [10] *Catalex_MP3_board.Pdf*. dirección: https://geekmatic.in.ua/pdf/Catalex_MP3_board.pdf (visitado 11-05-2024).
- [11] *M24SR64-Y - Pag web*. dirección: <https://www.st.com/en/nfc/m24sr64-y.html> (visitado 11-05-2024).
- [12] *NFC Tools*. dirección: <https://play.google.com/store/games?hl=es&gl=US> (visitado 11-05-2024).
- [13] *ST25*. dirección: <https://play.google.com/store/games?hl=es&gl=US> (visitado 11-05-2024).
- [14] *JustGage*. dirección: <https://plainjs.com/javascript/plugins/justgage-118/> (visitado 12-05-2024).
- [15] *Ping Pong Buffer Audio Stream*, jun. de 2020. dirección: <https://audiodsplab.wordpress.com/ping-pong-buffer-audio-stream/> (visitado 12-05-2024).
- [16] *CMSIS DSP Software Library*. dirección: https://arm-software.github.io/CMSIS_5/DSP/html/index.html (visitado 12-05-2024).
- [17] «Q (number format),» Wikipedia, abr. de 2024. dirección: [https://en.wikipedia.org/w/index.php?title=Q_\(number_format\)&oldid=1219039759](https://en.wikipedia.org/w/index.php?title=Q_(number_format)&oldid=1219039759) (visitado 12-05-2024).
- [18] *Graphic Audio Equalizer Example*. dirección: https://arm-software.github.io/CMSIS_5/DSP/html/group__GEQ5Band.html (visitado 12-05-2024).
- [19] *Cálculo CRC*. dirección: <https://hub.zhovner.com/tools/nfc/> (visitado 11-05-2024).
- [20] *FatFs Module Application Note*. dirección: <http://elm-chan.org/fsw/ff/doc/appnote.html> (visitado 11-05-2024).
- [21] *FatFs - Generic FAT Filesystem Module*. dirección: <http://elm-chan.org/fsw/ff/> (visitado 06-05-2024).

Anexo A Webench Design Report

WEBENCH® Design Report

Design : 1 LM51561HPWPR
LM51561HPWPR Boost Converter

VinMin = 3.5V
VinMax = 4.2V
Vout = 7.0V
Iout = 1.0A
Device = LM51561HPWPR
Topology = Boost
Created = 2024-04-02 12:11:39.716
BOM Cost = \$3.03
BOM Count = 22
Total Pd = 0.69W

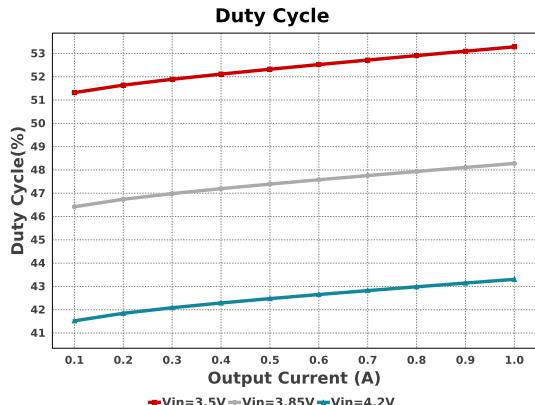
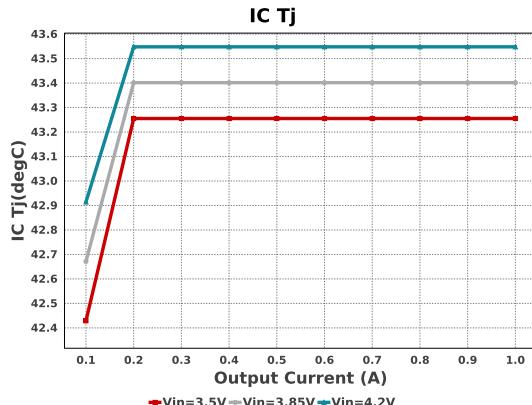


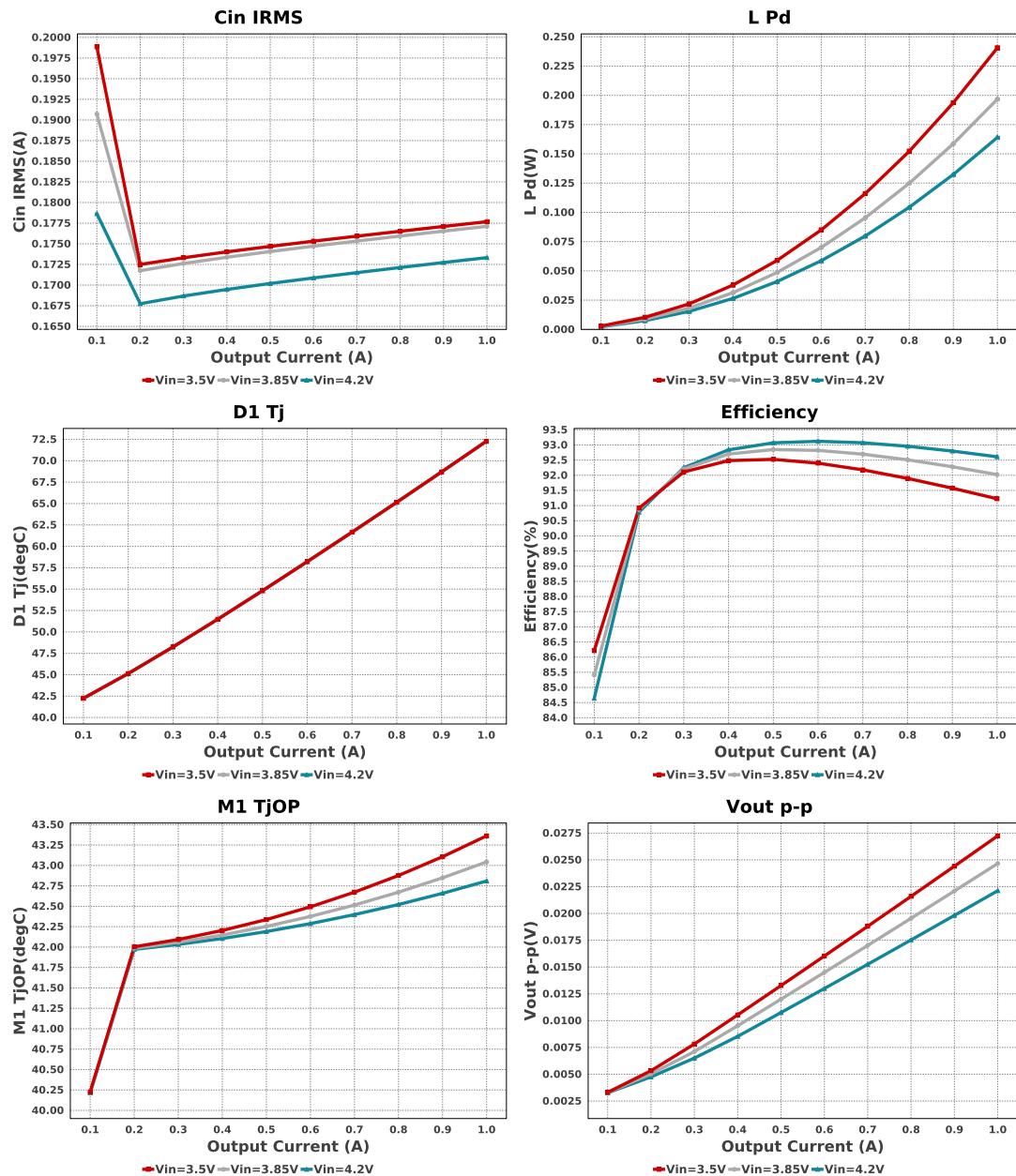
Electrical BOM

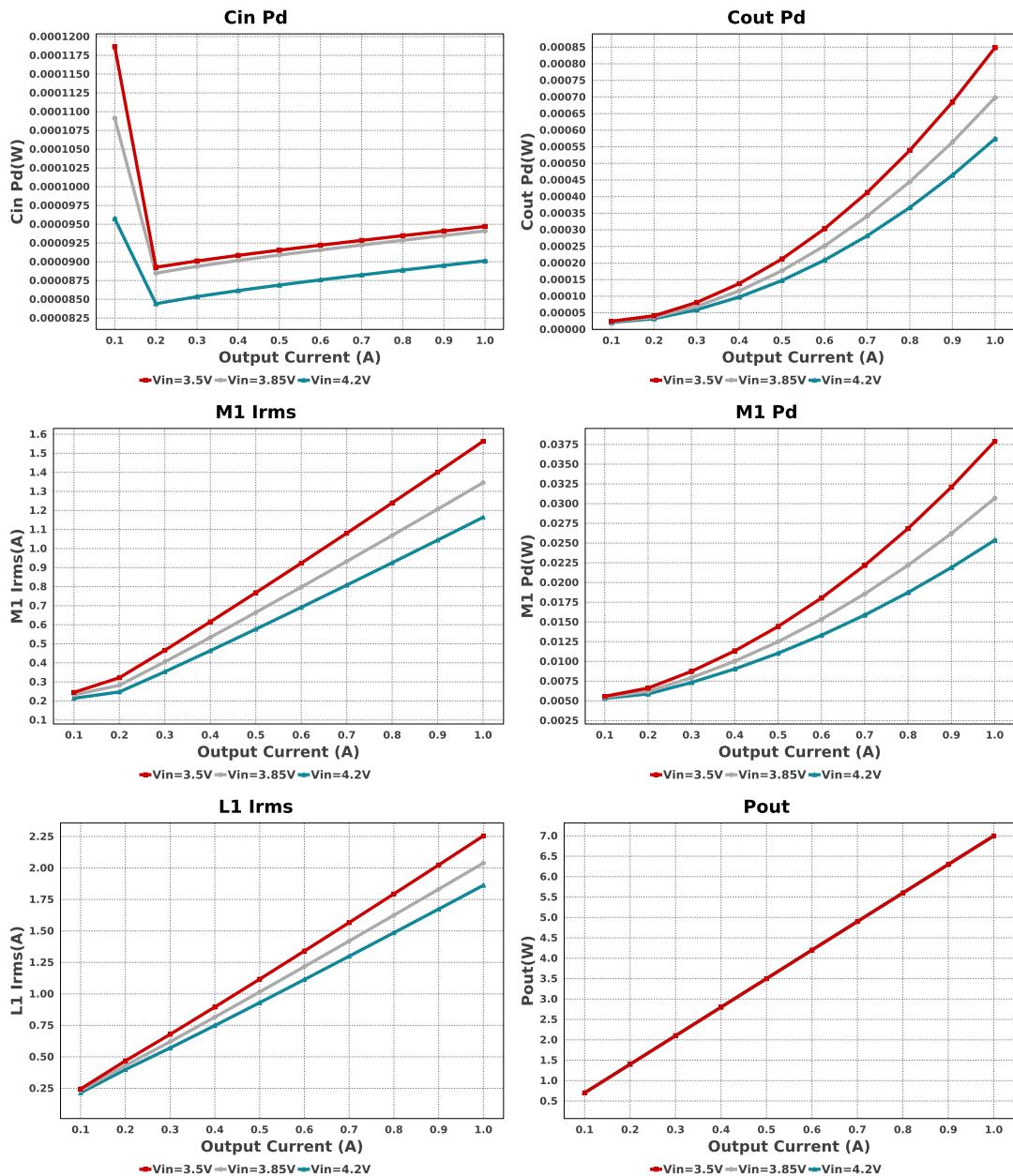
Name	Manufacturer	Part Number	Properties	Qty	Price	Footprint
Cbias	MuRata	GRM155R70J104KA01D Series= X7R	Cap= 100.0 nF ESR= 1.0 mOhm VDC= 6.3 V IRMS= 0.0 A	1	\$0.01	■ 0402 3 mm²
Ccomp	MuRata	GRM155R71E273KA88D Series= X7R	Cap= 27.0 nF ESR= 1.0 mOhm VDC= 25.0 V IRMS= 0.0 A	1	\$0.01	■ 0402 3 mm²
Ccomp2	MuRata	GRM155R71E562KA01D Series= X7R	Cap= 5.6 nF ESR= 1.0 mOhm VDC= 25.0 V IRMS= 0.0 A	1	\$0.01	■ 0402 3 mm²
Ccs	Samsung Electro-Mechanics	CL21C102JBCNNNC Series= C0G/NP0	Cap= 1.0 nF VDC= 50.0 V IRMS= 0.0 A	1	\$0.01	■ 0805 7 mm²
Cin	Kemet	C0805C106K8PACTU Series= X5R	Cap= 10.0 uF ESR= 3.0 mOhm VDC= 10.0 V IRMS= 11.43 A	1	\$0.03	■ 0805 7 mm²
Cout	MuRata	GRM32ER61E226KE15L Series= X5R	Cap= 22.0 uF ESR= 2.0 mOhm VDC= 25.0 V IRMS= 3.67 A	3	\$0.23	■ 1210 15 mm²
Coutx	TDK	C1005X6S1C105K050BC Series= X6S	Cap= 1.0 uF ESR= 11.416 mOhm VDC= 16.0 V IRMS= 1.483 A	1	\$0.02	■ 0402 3 mm²
Css	AVX	08053C104JAZ2A Series= X7R	Cap= 100.0 nF VDC= 25.0 V IRMS= 0.0 A	1	\$0.07	■ 0805 7 mm²
Cvcc	Taiyo Yuden	EMK107B7105KA-T Series= X7R	Cap= 1.0 uF ESR= 1.0 mOhm VDC= 16.0 V IRMS= 0.0 A	1	\$0.01	■ 0603 5 mm²
D1	Toshiba	CMS06	VF@Io= 320.0 mV VRRM= 30.0 V	1	\$0.20	■ M-FLAT 19 mm²

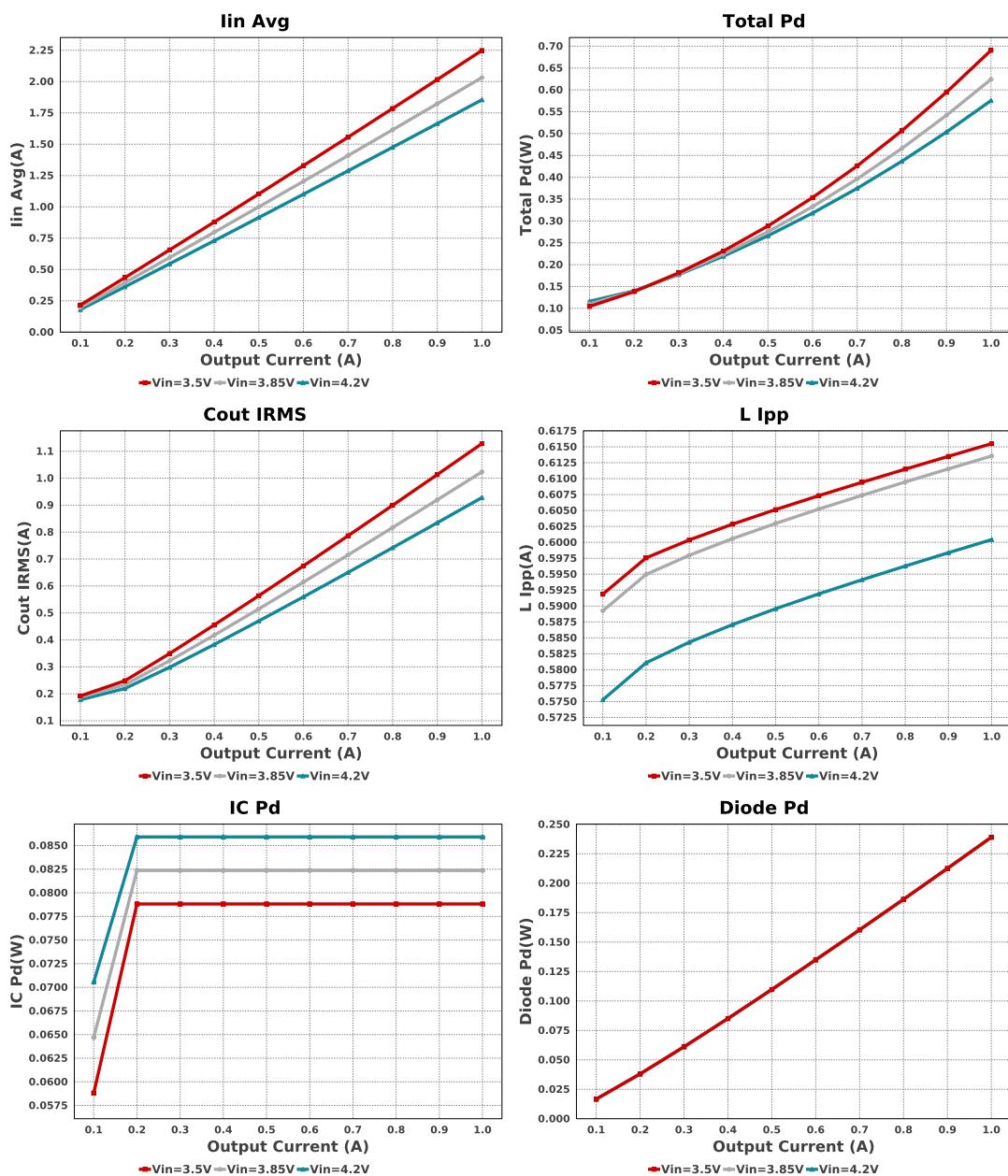
WEBENCH® Design

Name	Manufacturer	Part Number	Properties	Qty	Price	Footprint
L1	Bourns	SRN6045-6R8Y	L= 6.8 μ H 47.3 mOhm	1	\$0.25	 SRN6045 64 mm ²
M1	Fairchild Semiconductor	FDD8647L	VdsMax= 40.0 V IdsMax= 14.0 Amps	1	\$0.70	 DPAK 102 mm ²
Rcomp	Susumu Co Ltd	RG1608P-3571-B-T5 Series= RG1608	Res= 3.57 kOhm Power= 100.0 mW Tolerance= 0.1%	1	\$0.06	 0603 5 mm ²
Rcs	Stackpole Electronics Inc	CSR1206FK30L0 Series= ?	Res= 30.0 mOhm Power= 500.0 mW Tolerance= 1.0%	1	\$0.10	 1206 11 mm ²
Rcs1	Vishay-Dale	CRCW0402100RFKED Series= CRCW..e3	Res= 100.0 Ohm Power= 63.0 mW Tolerance= 1.0%	1	\$0.01	 0402 3 mm ²
Rfbb	Susumu Co Ltd	RG1608P-2491-B-T5 Series= RG1608	Res= 2.49 kOhm Power= 100.0 mW Tolerance= 0.1%	1	\$0.06	 0603 5 mm ²
Rfbt	Susumu Co Ltd	RG1608P-153-B-T5 Series= RG1608	Res= 15.0 kOhm Power= 100.0 mW Tolerance= 0.1%	1	\$0.04	 0603 5 mm ²
Rsl	Vishay-Dale	CRCW0402732RFKED Series= CRCW..e3	Res= 732.0 Ohm Power= 63.0 mW Tolerance= 1.0%	1	\$0.01	 0402 3 mm ²
Rt	Vishay-Dale	CRCW040248K7FKED Series= CRCW..e3	Res= 48.7 kOhm Power= 63.0 mW Tolerance= 1.0%	1	\$0.01	 0402 3 mm ²
U1	Texas Instruments	LM51561HPWPR	Switcher	1	\$0.73	 PWP0014H 59 mm ²

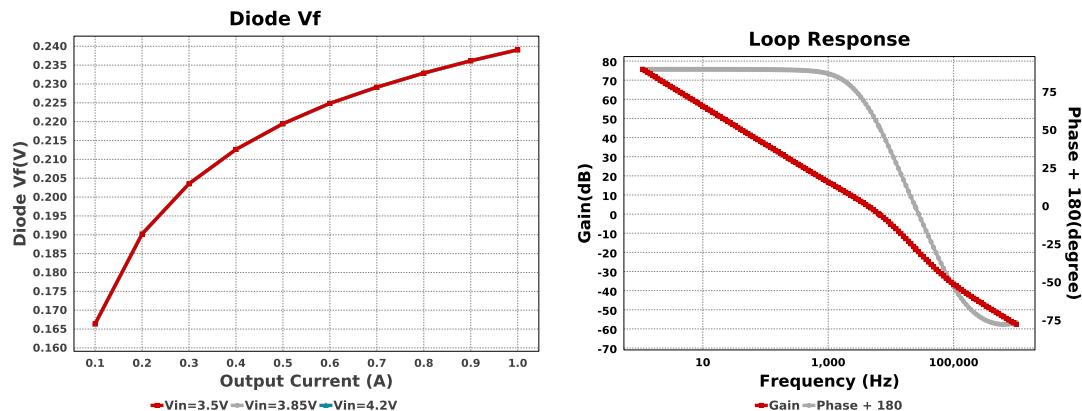


WEBENCH® Design

WEBENCH® Design

WEBENCH® Design

WEBENCH® Design



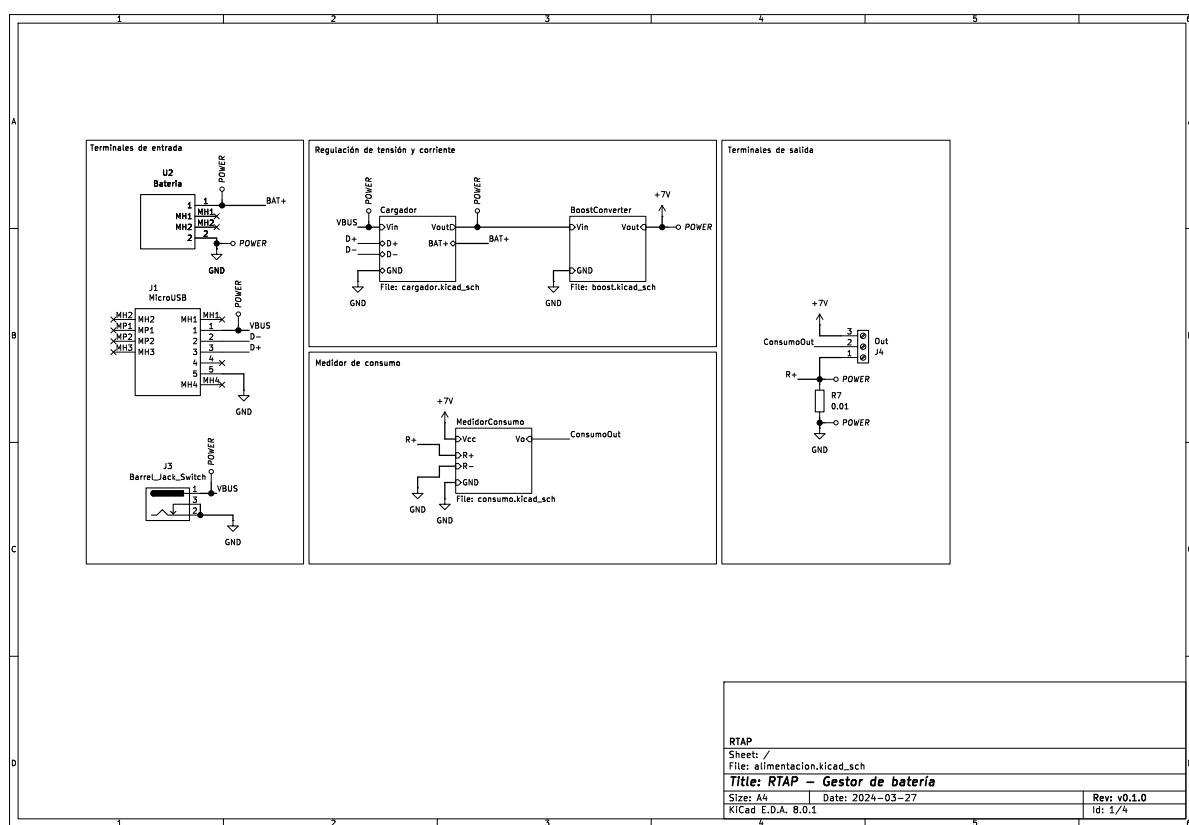
Operating Values

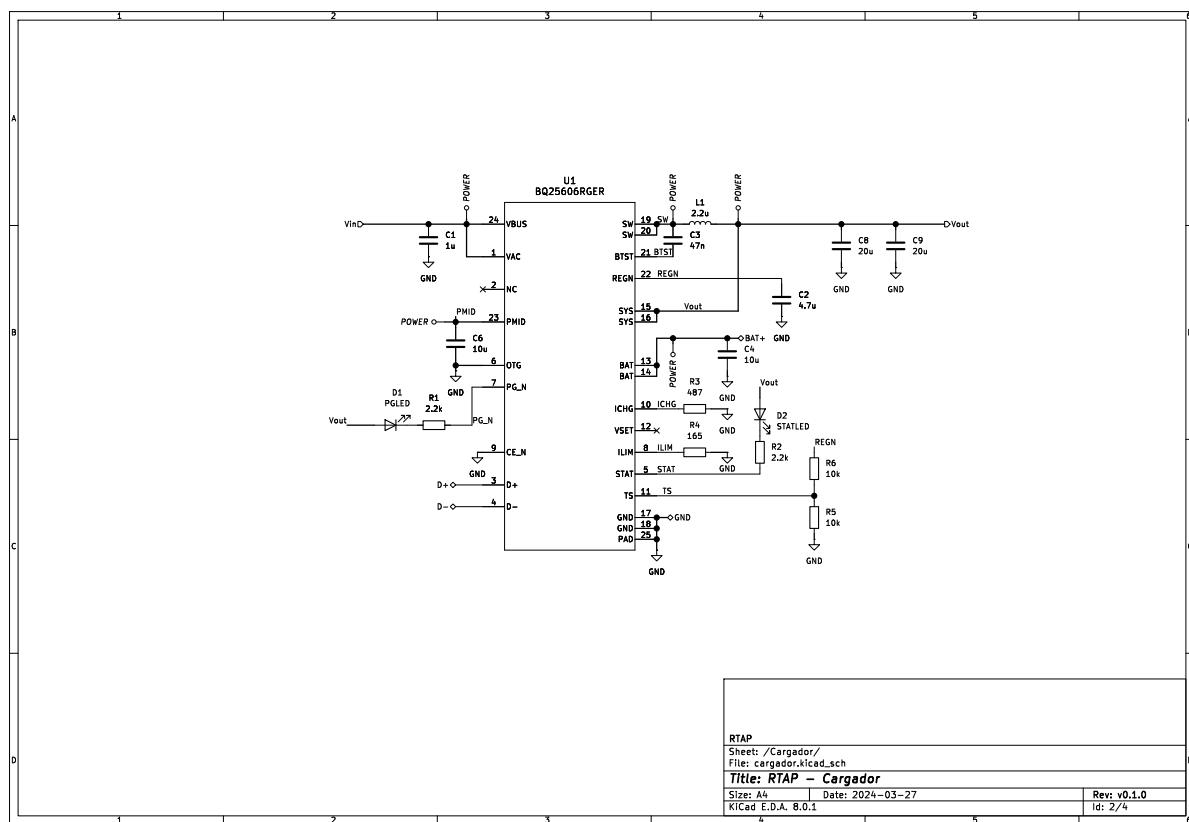
#	Name	Value	Category	Description
1.	BOM Count	22		Total Design BOM count
2.	Total BOM	\$3.03		Total BOM Cost
3.	Cin IRMS	177.679 mA	Capacitor	Input capacitor RMS ripple current
4.	Cin Pd	94.709 μ W	Capacitor	Input capacitor power dissipation
5.	Cout IRMS	1.128 A	Capacitor	Output capacitor RMS ripple current
6.	Cout Pd	848.63 μ W	Capacitor	Output capacitor power dissipation
7.	D1 Tj	72.272 degC	Diode	D1 junction temperature
8.	Diode Pd	239.05 mW	Diode	Diode power dissipation
9.	Diode Vf	239.053 mV	Diode	Forward voltage drop of diode D1
10.	IC Pd	78.819 mW	IC	IC power dissipation
11.	IC Tj	43.255 degC	IC	IC junction temperature
12.	ICThetaJA	41.3 degC/W	IC	IC junction-to-ambient thermal resistance
13.	Iin Avg	2.248 A	IC	Average input current
14.	L Ipp	615.498 mA	Inductor	Peak-to-peak inductor ripple current
15.	L Pd	240.58 mW	Inductor	Inductor power dissipation
16.	L1 Irms	2.255 A	Inductor	Inductor ripple current
17.	M1 Irms	1.563 A	Mosfet	Q lavg
18.	M1 Pd	37.898 mW	Mosfet	MOSFET power dissipation
19.	M1 TjOP	43.363 degC	Mosfet	M1 MOSFET junction temperature
20.	Cin Pd	94.709 μ W	Power	Input capacitor power dissipation
21.	Cout Pd	848.63 μ W	Power	Output capacitor power dissipation
22.	Diode Pd	239.05 mW	Power	Diode power dissipation
23.	IC Pd	78.819 mW	Power	IC power dissipation
24.	L Pd	240.58 mW	Power	Inductor power dissipation
25.	M1 Pd	37.898 mW	Power	MOSFET power dissipation
26.	Total Pd	690.478 mW	Power	Total Power Dissipation
27.	Cross Freq	5.588 kHz	System Information	Bode plot crossover frequency
28.	Duty Cycle	53.284 %	System Information	Duty cycle
29.	Efficiency	91.225 %	System Information	Steady state efficiency
30.	FootPrint	359.0 mm ²	System Information	Total Foot Print Area of BOM components
31.	Frequency	444.713 kHz	System Information	Switching frequency
32.	Iout	1.0 A	System Information	Iout operating point
33.	Mode	CCM	System Information	Conduction Mode
34.	Phase Marg	54.932 deg	System Information	Bode Plot Phase Margin
35.	Pout	7.0 W	System Information	Total output power
36.	Vin	3.5 V	System Information	Vin operating point
37.	Vout Actual	7.024 V	System Information	Vout Actual calculated based on selected voltage divider resistors
38.	Vout Tolerance	1.173 %	System Information	Vout Tolerance based on IC Tolerance (no load) and voltage divider resistors if applicable
39.	Vout p-p	27.241 mV	System Information	Peak-to-peak output ripple voltage

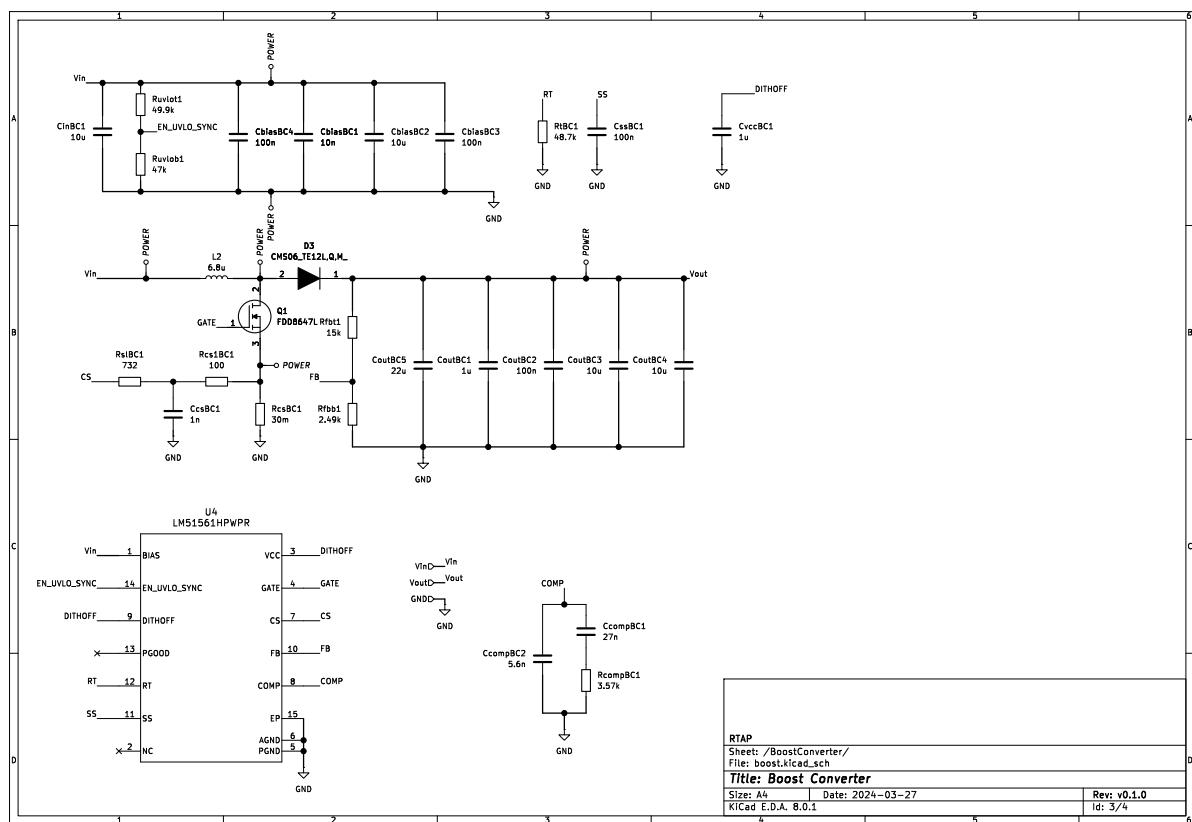
WEBENCH® Design**Design Inputs**

Name	Value	Description
Iout	1.0	Maximum Output Current
VinMax	4.2	Maximum input voltage
VinMin	3.5	Minimum input voltage
Vout	7.0	Output Voltage
base_pn	LM51561H	Base Product Number
source	DC	Input Source Type
Ta	40.0	Ambient temperature

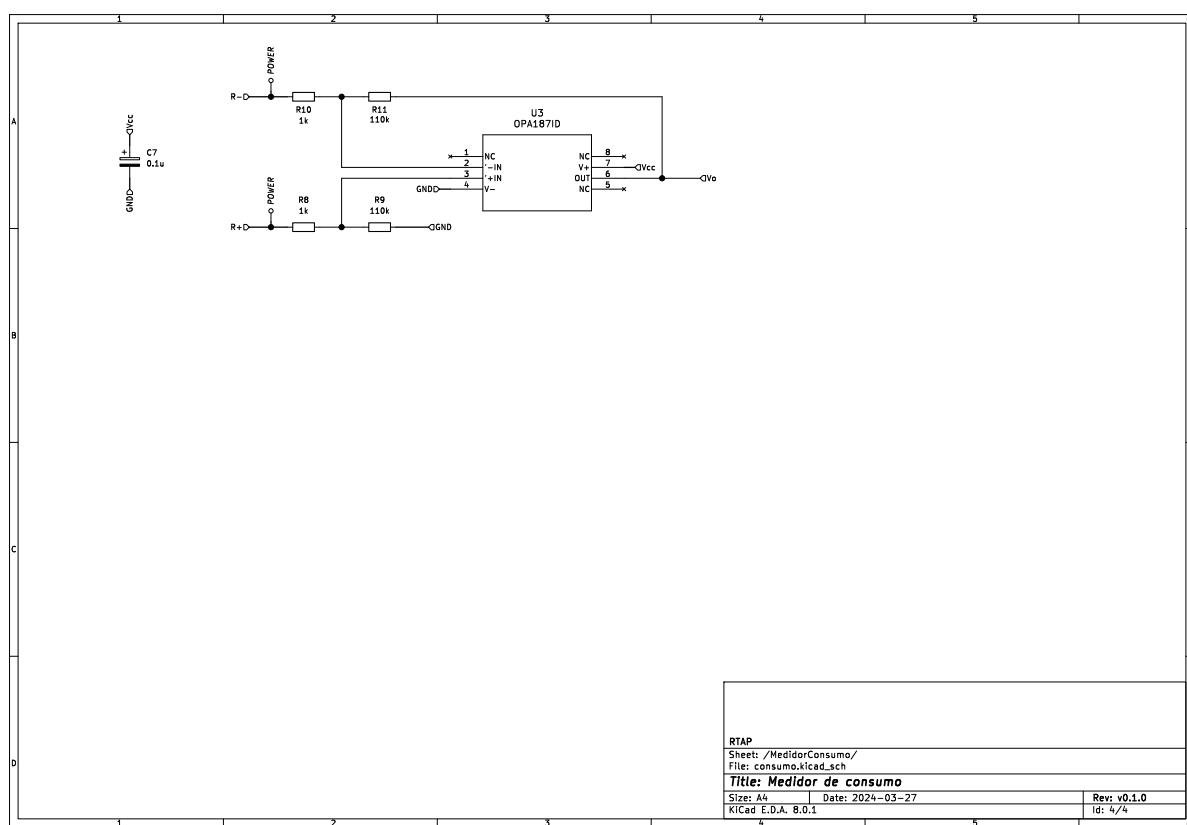
Anexo B Circuito de alimentación







Anexo C Circuito de alimentación



Anexo D Estructura de los mensajes de control

Fragmento 8: Fichero controlThread.h con las estructuras de mensajes

```

/***
* @file controlThread.h
*
* @brief Modulo de control de RTAP
*
* @author Ruben Agustin
* @author David Andriño
* @author Estela Mora
* @author Fernando Sanz
*
* Modulo principal de inteligencia del sistema. Recibe eventos por
* una cola y reacciona acorde a ellos
*
*/
#ifndef CONTROL_THREAD_H
#define CONTROL_THREAD_H

#include <cmsis_os2.h>
#include <stdint.h>

#include "../main.h"
#include "../SD/sd.h"

/***
* @brief Cola de mensajes de entrada al modulo
*/
extern osMessageQueueId_t ctrl_in_queue;

/***
* @brief Inicializacion del modulo de control
*
* @return 0 si se ha realizado correctamente. Otro valor si no.
*/
int Init_Control(sd_config_t* initial_config);

// ===== MSG TYPES =====
/***
* @brief Enumeracion de los tipos de mensaje de entrada al modulo de control
*/
typedef enum {
    MSG_NFC,    /**< Lectura de una tarjeta del NFC */
    MSG_LCD,    /**< Mensaje de entrada del LCD */
    MSG_WEB,    /**< Mensaje de entrada de la web */
    MSG_RTC,    /**< Mensaje de entrada del RTC */
    MSG_CONS,   /**< Mensaje de entrada del consumo */
    MSG_RADIO,  /**< Mensaje de entrada de la radio */
} msg_ctrl_type_t;

// ===== LCD =====
/***
* @brief Enumeracion de mensajes de entrada del LCD
*/
typedef enum {
    LCD_VOL,      /**< Cambio de volumen. Contenido es el volumen */
    LCD_BANDS,    /**< Cambio de filtro. Contenido es primer byte la banda [0,4] segundo la
                   cantidad [-9, 9] */
    LCD_RADIO_FREQ, /**< Cambio de frecuencia de la radio. Contenido es la frecuencia en
                     centenas de kHz */
}

```

```

LCD_SONG,      /**< Cambio de cancion. Contenido es el numero de cancion, empezando por
               la 0 */
LCD_INPUT_SEL, /**< Cambio de entrada. Contenido es 0 para la radio y 1 para MP3 */
LCD_OUTPUT_SEL, /**< Cambio de salida. Contenido es 0 para cascos y 1 para altavoz */
LCD_SAVE_SD,   /**< Guardar configuracion en la SD. Contenido ignorado */
LCD_LOW_POWER, /**< Entrar en modo bajo consumo. Contenido ignorado */
LCD_LOOP,      /**< Poner cancion en bucle. Contenido ignorado*/
LCD_SEEK,      /**< Hacer seek con la radio. Contenido es 0 para down y 1 para up */
LCD_NEXT_SONG, /**< Siguiente cancion */
LCD_PREV_SONG, /**< Anterior cancion */
LCD_PLAY_PAUSE, /**< Alternar reproduccion de la cancion */
} lcd_msg_type_t;

/**
* @brief Estructura para los mensajes de entrada del LCD
*/
typedef struct {
    lcd_msg_type_t type; /**< Tipo de mensaje del LCD */
    uint16_t payload; /**< Contenido del mensaje. Depende del tipo. */
} lcd_msg_t;

// ===== WEB =====
/**
* @brief Enumeracion de mensajes de entrada de la web
*/
typedef enum {
    WEB_INPUT_SEL, /**< Cambio de entrada. Contenido es 0 para la radio y 1 para MP3 */
    WEB_OUTPUT_SEL, /**< Cambio de salida. Contenido es 0 para cascos y 1 para altavoz */
    WEB_LOW_POWER, /**< Entrar en modo bajo consumo. Contenido ignorado */
    WEB_RADIO_FREQ, /**< Cambio de frecuencia de la radio. Contenido es la frecuencia en
                      centenas de kHz */
    WEB_SEEK,      /**< Hacer seek con la radio. Contenido es 0 para down y 1 para up */
    WEB_VOL,       /**< Cambio de volumen. Contenido es el volumen */
    WEB_SONG,      /**< Cambio de cancion. Contenido es el numero de cancion */
    WEB_PLAY_PAUSE, /**< Alternar play y pause de la web */
    WEB_PREV_SONG, /**< Anterior cancion */
    WEB_NEXT_SONG, /**< Siguiente cancion*/
    WEB_BANDS,     /**< Cambio de filtro. Contenido es primer byte la banda [0,4] segundo la
                    cantidad [-9, 9] */
    WEB_SAVE_SD,   /**< Guardar configuracion en la SD. Contenido ignorado */
    WEB_LOOP,      /**< Poner cancion en bucle. Contenido ignorado*/
} web_msg_type_t;

/**
* @brief Estructura para los mensajes de entrada del LCD
*/
typedef struct {
    web_msg_type_t type; /**< Tipo del mensaje de */
    uint16_t payload; /**< Contenido del mensaje. Depende del tipo */
} web_msg_t;

// ===== NFC =====
/**
* @brief Estructura para los mensajes de entrada del NFC
*/
typedef struct {
    uint8_t type;   /**< Tipo de mensaje. 0 para cancion y 1 para radio*/
    uint17_t content; /**< Numero de cancion o frecuencia en centenas*/
} nfc_msg_t;

// ===== RTC =====
/**
* @brief Estructura para los mensajes de entrada del RTC

```

```

/*
typedef struct {
    uint8_t hour, minute, second, day, month, year;
} rtc_msg_t;

// ===== MSG =====
/** 
 * @brief Estructura para los mensajes de entrada
 */
typedef struct {
    msg_ctrl_type_t type; /*< Tipo de mensaje de entrada.
                           Dependiendo de este valor se debe interpretar el contenido */
    union {
        nfc_msg_t nfc_msg; /*< Contenido de un mensaje de tipo MSG_NFC */
        lcd_msg_t lcd_msg; /*< Contenido de un mensaje de tipo MSG_LCD */
        rtc_msg_t rtc_msg; /*< Contenido de un mensaje de tipo MSG_RTC */
        web_msg_t web_msg; /*< Contenido de un mensaje de tipo MSG_WEB */
        uint16_t cons_msg; /*< Contenido de un mensaje de tipo MSG_CONS */
        uint32_t radio_msg; /*< Contenido de un mensaje de tipo MSG_RADIO */
    };
} msg_ctrl_t;

#endif

```

Anexo E Ficheros de adaptación de la pantalla

Fragmento 9: Fichero tft.h

```

/**
* @file disp.h
*
*/
#ifndef DISP_H
#define DISP_H

/*****
*      INCLUDES
*****
#include <stdint.h>
#include "lvgl.h"

/*****
*      DEFINES
*****
#define TFT_HOR_RES 800
#define TFT_VER_RES 480
#define TFT_NO_TEARING 0 /*1: no tearing but slower*/

/*****
*      TYPEDEFS
*****
/*****
* GLOBAL PROTOTYPES
*****
void tft_init(void);
/*****
*      MACROS
*****

```

```
#endif
```

Fragmento 10: Fichero tft.c

```
/*
 * @file disp.c
 *
 */

/*****
 *      INCLUDES
 *****/
#include "../lv_conf.h"
#include "lvgl.h"
#include <string.h>

#include "tft.h"
#include "stm32f7xx_hal.h"

#include "stm32f769i_discovery.h"
#include "stm32f769i_discovery_lcd.h"
#include "stm32f769i_discovery_sdram.h"

/*****
 *      DEFINES
 ****/
/* DMA Stream parameters definitions. You can modify these parameters to select
   a different DMA Stream and/or channel.
   But note that only DMA2 Streams are capable of Memory to Memory transfers. */
#define DMA_STREAM          DMA2_Stream2
#define DMA_CHANNEL         DMA_CHANNEL_2
#define DMA_STREAM_IRQ      DMA2_Stream2_IRQn
#define DMA_STREAM_IRQHandler DMA2_Stream2_IRQHandler

#if TFT_NO_TEARING
#define ZONES              4      /*Divide the screen into zones to handle tearing effect*/
#else
#define ZONES              1
#endif

#define VSYNC               OTM8009A_800X480_VSYNC
#define VBP                 OTM8009A_800X480_VBP
#define VFP                 OTM8009A_800X480_VFP
#define VACT                OTM8009A_800X480_HEIGHT
#define HSYNC               OTM8009A_800X480_HSYNC
#define HBP                 OTM8009A_800X480_HBP
#define HFP                 OTM8009A_800X480_HFP
#define HACT                (OTM8009A_800X480_WIDTH / ZONES)

#define LAYER0_ADDRESS       (LCD_FB_START_ADDRESS)

/*****
 *      TYPEDEFS
 ****/
/*****
 *  STATIC PROTOTYPES
 ****/
/*For LittlevGL*/
static void tft_flush_cb(lv_disp_t * disp, const lv_area_t * area, uint8_t * pxmap);
```

```

/*LCD*/
static void LCD_Config(void);
static void LTDC_Init(void);

/*DMA to flush to frame buffer*/
static void DMA_Config(void);
static void DMA_TransferComplete(DMA_HandleTypeDef *han);
static void DMA_TransferError(DMA_HandleTypeDef *han);

/******************
 * STATIC VARIABLES
 *****************/
extern LTDC_HandleTypeDef hltdc_discovery;
extern DSI_HandleTypeDef hdsi_discovery;
DSI_VidCfgTypeDef hdsivideo_handle;
DSI_CmdCfgTypeDef CmdCfg;
DSI_LPCmdTypeDef LPCmd;
DSI_PLLInitTypeDef dsiPllInit;
static RCC_PeriphCLKInitTypeDef PeriphClkInitStruct;

#if LV_COLOR_DEPTH == 16
static uint16_t * my_fb = (uint16_t *)LAYER0_ADDRESS;
#else
static uint32_t * my_fb = (uint32_t *)LAYER0_ADDRESS;
#endif

lv_display_t * disp;

static DMA_HandleTypeDef DmaHandle;
static volatile int32_t x1_flush;
static volatile int32_t y1_flush;
static volatile int32_t x2_flush;
static volatile int32_t y2_flush;
static volatile int32_t y_flush_act;
static volatile const uint8_t * buf_to_flush;

static volatile bool refr_qry;
static volatile uint32_t t_last = 0;

#if TFT_NO_TEARING
uint8_t pPage[] = {0x00, 0x00, 0x01, 0xDF}; /* 0 -> 479 */
#endif

uint8_t pCols[ZONEs][4] =
{
#if (ZONEs == 4 )
    {0x00, 0x00, 0x00, 0xC7}, /* 0 -> 199 */
    {0x00, 0xC8, 0x01, 0x8F}, /* 200 -> 399 */
    {0x01, 0x90, 0x02, 0x57}, /* 400 -> 599 */
    {0x02, 0x58, 0x03, 0x1F}, /* 600 -> 799 */
#elif (ZONEs == 2 )
    {0x00, 0x00, 0x01, 0x8F}, /* 0 -> 399 */
    {0x01, 0x90, 0x03, 0x1F}
#elif (ZONEs == 1 )
    {0x00, 0x00, 0x03, 0x1F}, /* 0 -> 799 */
#endif
};

/******************
 * MACROS
 *****************/

```

```
*****
/***** GLOBAL FUNCTIONS *****/
*****

/***
 * Initialize your display here
 */
void tft_init(void)
{
    BSP_SDRAM_Init();
    /* Deactivate speculative/cache access to first FMC Bank to save FMC bandwidth */
    FMC_Bank1->BTCR[0] = 0x000030D2;
    LCD_Config();

    /* Send Display On DCS Command to display */
    HAL_DSI_ShortWrite(&(hdsi_discovery),
        0,
        DSI_DCS_SHORT_PKT_WRITE_P1,
        OTM8009A_CMD_DISPON,
        0x00);

    DMA_Config();

    static uint8_t buf1[TFT_HOR_RES * 48 * 2];
    static uint8_t buf2[TFT_HOR_RES * 48 * 2];
    disp = lv_display_create(800, 480);
    lv_display_set_buffers(disp, buf1, buf2, TFT_HOR_RES * 48 * 2,
        LV_DISP_RENDER_MODE_PARTIAL);
    lv_display_set_flush_cb(disp, tft_flush_cb);
}

/***** STATIC FUNCTIONS ****/
*****
```

```
static void tft_flush_cb(lv_disp_t * disp, const lv_area_t * area, uint8_t * pxmap)
{
    SCB_CleanInvalidateDCache();

    /*Truncate the area to the screen*/
    int32_t act_x1 = area->x1 < 0 ? 0 : area->x1;
    int32_t act_y1 = area->y1 < 0 ? 0 : area->y1;
    int32_t act_x2 = area->x2 > TFT_HOR_RES - 1 ? TFT_HOR_RES - 1 : area->x2;
    int32_t act_y2 = area->y2 > TFT_VER_RES - 1 ? TFT_VER_RES - 1 : area->y2;

    x1_flush = act_x1;
    y1_flush = act_y1;
    x2_flush = act_x2;
    y2_flush = act_y2;
    y_flush_act = act_y1;
    buf_to_flush = pxmap;

    /*Use DMA instead of DMA2D to leave it free for GPU*/
    HAL_StatusTypeDef err;
    err = HAL_DMA_Start_IT(&DmaHandle,(uint32_t)buf_to_flush, (uint32_t)&my_fb[y_flush_act * TFT_HOR_RES + x1_flush],
        (x2_flush - x1_flush + 1));
    if(err != HAL_OK)
    {
        while(1); /*Halt on error*/
    }
}
```

```

    }

}

static void LCD_Config(void)
{
    DSI_PHY_TimerTypeDef PhyTimings;

    /* Toggle Hardware Reset of the DSI LCD using
     * its XRES signal (active low) */
    BSP_LCD_Reset();

    /* Call first MSP Initialize only in case of first initialization
     * This will set IP blocks LTDC, DSI and DMA2D
     * - out of reset
     * - clocked
     * - NVIC IRQ related to IP blocks enabled
     */
    BSP_LCD_MspInit();

    /* LCD clock configuration */
    /* PLLSAI_VCO Input = HSE_VALUE/PLL_M = 1 Mhz */
    /* PLLSAI_VCO Output = PLLSAI_VCO Input * PLLSAIN = 417 Mhz */
    /* PLLCDCLK = PLLSAI_VCO Output/PLLSAIR = 417 MHz / 5 = 83.4 MHz */
    /* LTDC clock frequency = PLLCDCLK / LTDC_PLLSAI_DIVR_2 = 83.4 / 2 = 41.7 MHz */
    PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_LTDC;
    PeriphClkInitStruct.PLLSAI.PLLSAIN = 417;
    PeriphClkInitStruct.PLLSAI.PLLSAIR = 5;
    PeriphClkInitStruct.PLLSAIDivR = RCC_PLLSAIDIVR_2;
    HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct);

    /* Base address of DSI Host/Wrapper registers to be set before calling De-Init */
    hdsi_discovery.Instance = DSI;

    HAL_DSI_DeInit(&(hdsi_discovery));

    dsiPllInit.PLLNDIV = 100;
    dsiPllInit.PLLIDF = DSI_PLL_IN_DIV5;
    dsiPllInit.PLLODF = DSI_PLL_OUT_DIV1;

    hdsi_discovery.Init.NumberOfLanes = DSI_TWO_DATA_LANES;
    hdsi_discovery.Init.TXEscapeCkdiv = 0x4;
    HAL_DSI_Init(&(hdsi_discovery), &(dsiPllInit));

    /* Configure the DSI for Command mode */
    CmdCfg.VirtualChannelID      = 0;
    CmdCfg.HSPolarity           = DSI_HSYNC_ACTIVE_HIGH;
    CmdCfg.VSPolarity           = DSI_VSYNC_ACTIVE_HIGH;
    CmdCfg.DEPolarity            = DSI_DATA_ENABLE_ACTIVE_HIGH;
#if LV_COLOR_DEPTH == 16
    CmdCfg.ColorCoding          = DSI_RGB565;
#else
    CmdCfg.ColorCoding          = DSI_RGB888;
#endif
    CmdCfg.CommandSize           = HACT;
    CmdCfg.TearingEffectSource  = DSI_TE_EXTERNAL;
    CmdCfg.TearingEffectPolarity = DSI_TE_RISING_EDGE;
    CmdCfg.VSyncPol              = DSI_VSYNC_FALLING;
    CmdCfg.AutomaticRefresh     = DSI_AR_DISABLE;
    CmdCfg.TEAcknowledgeRequest = DSI_TE_ACKNOWLEDGE_ENABLE;
    HAL_DSI_ConfigAdaptedCommandMode(&hdsi_discovery, &CmdCfg);

    LPCmd.LPGenShortWriteNoP = DSI_LP_GSWOP_ENABLE;
    LPCmd.LPGenShortWriteOneP = DSI_LP_GSW1P_ENABLE;
}

```

```

LPCmd.LPGenShortWriteTwoP = DSI_LP_GSW2P_ENABLE;
LPCmd.LPGenShortReadNoP = DSI_LP_GSR0P_ENABLE;
LPCmd.LPGenShortReadOneP = DSI_LP_GSR1P_ENABLE;
LPCmd.LPGenShortReadTwoP = DSI_LP_GSR2P_ENABLE;
LPCmd.LPGenLongWrite = DSI_LP_GLW_ENABLE;
LPCmd.LPDcsShortWriteNoP = DSI_LP_DSW0P_ENABLE;
LPCmd.LPDcsShortWriteOneP = DSI_LP_DSW1P_ENABLE;
LPCmd.LPDcsShortReadNoP = DSI_LP_DSROP_ENABLE;
LPCmd.LPDcsLongWrite = DSI_LP_DLW_ENABLE;
HAL_DSI_ConfigCommand(&hdsi_discovery, &LPCmd);

/* Initialize LTDC */
LTDC_Init();

/* Start DSI */
HAL_DSI_Start(&(hdsi_discovery));

/* Configure DSI PHY HS2LP and LP2HS timings */
PhyTimings.ClockLaneHS2LPTime = 35;
PhyTimings.ClockLaneLP2HSTime = 35;
PhyTimings.DataLaneHS2LPTime = 35;
PhyTimings.DataLaneLP2HSTime = 35;
PhyTimings.DataLaneMaxReadTime = 0;
PhyTimings.StopWaitTime = 10;
HAL_DSI_ConfigPhyTimer(&hdsi_discovery, &PhyTimings);

/* Initialize the OTM8009A LCD Display IC Driver (KoD LCD IC Driver)
 * depending on configuration set in 'hdsivideo_handle'.
 */
#endif
#if LV_COLOR_DEPTH == 16
OTM8009A_Init(OTM8009A_FORMAT_RGB565, LCD_ORIENTATION_LANDSCAPE);
#else
OTM8009A_Init(OTM8009A_FORMAT_RGB888, LCD_ORIENTATION_LANDSCAPE);
#endif
LPCmd.LPGenShortWriteNoP = DSI_LP_GSW0P_DISABLE;
LPCmd.LPGenShortWriteOneP = DSI_LP_GSW1P_DISABLE;
LPCmd.LPGenShortWriteTwoP = DSI_LP_GSW2P_DISABLE;
LPCmd.LPGenShortReadNoP = DSI_LP_GSR0P_DISABLE;
LPCmd.LPGenShortReadOneP = DSI_LP_GSR1P_DISABLE;
LPCmd.LPGenShortReadTwoP = DSI_LP_GSR2P_DISABLE;
LPCmd.LPGenLongWrite = DSI_LP_GLW_DISABLE;
LPCmd.LPDcsShortWriteNoP = DSI_LP_DSW0P_DISABLE;
LPCmd.LPDcsShortWriteOneP = DSI_LP_DSW1P_DISABLE;
LPCmd.LPDcsShortReadNoP = DSI_LP_DSROP_DISABLE;
LPCmd.LPDcsLongWrite = DSI_LP_DLW_DISABLE;
HAL_DSI_ConfigCommand(&hdsi_discovery, &LPCmd);

HAL_DSI_ConfigFlowControl(&hdsi_discovery, DSI_FLOW_CONTROL_BTA);

/* Send Display Off DCS Command to display */
HAL_DSI_ShortWrite(&(hdsi_discovery),
    0,
    DSI_DCS_SHORT_PKT_WRITE_P1,
    OTM8009A_CMD_DISPOFF,
    0x00);

#endif TFT_NO_TEARING
    HAL_DSI_LongWrite(&hdsi_discovery, 0, DSI_DCS_LONG_PKT_WRITE, 4, OTM8009A_CMD_CASET,
        pCols[0]);
    HAL_DSI_LongWrite(&hdsi_discovery, 0, DSI_DCS_LONG_PKT_WRITE, 4, OTM8009A_CMD_PASET,
        pPage);

```

```

/* Enable GPIOJ clock */
__HAL_RCC_GPIOJ_CLK_ENABLE();

/* Configure DSI_TE pin from MB1166 : Tearing effect on separated GPIO from KoD LCD */
/* that is mapped on GPIOJ2 as alternate DSI function (DSI_TE) */
/* This pin is used only when the LCD and DSI link is configured in command mode */
/* Not used in DSI Video mode.
 */
GPIO_InitTypeDef GPIO_Init_Structure;
GPIO_Init_Structure.Pin      = GPIO_PIN_2;
GPIO_Init_Structure.Mode    = GPIO_MODE_AF_PP;
GPIO_Init_Structure.Pull   = GPIO_NOPULL;
GPIO_Init_Structure.Speed  = GPIO_SPEED_HIGH;
GPIO_Init_Structure.Alternate = GPIO_AF13_DSI;
HAL_GPIO_Init(GPIOJ, &GPIO_Init_Structure);

    static uint8_t ScanLineParams[2];
#if ZONES == 2
    uint16_t scanline = 200;
#elif ZONES == 4
    uint16_t scanline = 283;
#endif
    ScanLineParams[0] = scanline >> 8;
    ScanLineParams[1] = scanline & 0x00FF;

    HAL_DSI_LongWrite(&hdsi_discovery, 0, DSI_DCS_LONG_PKT_WRITE, 2, 0x44, ScanLineParams);
    /* set_tear_on */
    HAL_DSI_ShortWrite(&hdsi_discovery, 0, DSI_DCS_SHORT_PKT_WRITE_P1, OTM8009A_CMD_TEEON,
                      0x00);
#endif
}

#if TFT_NO_TEARING
/**
 * LCD_SetUpdateRegion
 */
void LCD_SetUpdateRegion(int idx)
{
    HAL_DSI_LongWrite(&hdsi_discovery, 0, DSI_DCS_LONG_PKT_WRITE, 4, OTM8009A_CMD_CASET,
                      pCols[idx]);
}
#endif
/** 
 * @brief
 * @param None
 * @retval None
 */
static void LTDC_Init(void)
{
    /* DeInit */
    HAL_LTDC_DeInit(&hltdc_discovery);

    /* LTDC Config */
    /* Timing and polarity */
    hltdc_discovery.Init.HorizontalSync = HSYNC;
    hltdc_discovery.Init.VerticalSync = VSYNC;
    hltdc_discovery.Init.AccumulatedHBP = HSYNC+HBP;
    hltdc_discovery.Init.AccumulatedVBP = VSYNC+VBP;
    hltdc_discovery.Init.AccumulatedActiveH = VSYNC+VBP+VACT;
    hltdc_discovery.Init.AccumulatedActiveW = HSYNC+HBP+HACT;
    hltdc_discovery.Init.TotalHeigh = VSYNC+VBP+VACT+VFP;
    hltdc_discovery.Init.TotalWidth = HSYNC+HBP+HACT+HFP;
}

```

```

/* background value */
hltdc_discovery.Init.Backcolor.Blue = 0;
hltdc_discovery.Init.Backcolor.Green = 0;
hltdc_discovery.Init.Backcolor.Red = 0;

/* Polarity */
hltdc_discovery.Init.HSPolarity = LTDC_HSPOLARITY_AL;
hltdc_discovery.Init.VSPolarity = LTDC_VSPOLARITY_AL;
hltdc_discovery.Init.DEPolarity = LTDC_DEPOLARITY_AL;
hltdc_discovery.Init.PCPolarity = LTDC_PCPOLARITY_IPC;
hltdc_discovery.Instance = LTDC;

HAL_LTDC_Init(&hltdc_discovery);

LCD_LayerCfgTypeDef Layercfg;

/* Layer Init */
Layercfg.WindowX0 = 0;
Layercfg.WindowX1 = HACT;
Layercfg.WindowY0 = 0;
Layercfg.WindowY1 = BSP_LCD_GetYSize();
#if LV_COLOR_DEPTH == 16
Layercfg.PixelFormat = LTDC_PIXEL_FORMAT_RGB565;
#else
Layercfg.PixelFormat = LTDC_PIXEL_FORMAT_ARGB8888;
#endif
Layercfg.FBStartAddress = LAYER0_ADDRESS;
Layercfg.Alpha = 255;
Layercfg.Alpha0 = 0;
Layercfg.Backcolor.Blue = 0;
Layercfg.Backcolor.Green = 0;
Layercfg.Backcolor.Red = 0;
Layercfg.BlendingFactor1 = LTDC_BLENDING_FACTOR1_PAxCA;
Layercfg.BlendingFactor2 = LTDC_BLENDING_FACTOR2_PAxCA;
Layercfg.ImageWidth = BSP_LCD_GetXSize();
Layercfg.ImageHeight = BSP_LCD_GetYSize();

HAL_LTDC_ConfigLayer(&hltdc_discovery, &Layercfg, 0);

}

#if TFT_NO_TEARING
static volatile uint32_t LCD_ActiveRegion;

/**
 * @brief Tearing Effect DSI callback.
 * @param hdsi: pointer to a DSI_HandleTypeDef structure that contains
 *              the configuration information for the DSI.
 * @retval None
 */
void HAL_DSI_TearingEffectCallback(DSI_HandleTypeDef *hdsi)
{
    if(refr_qry) {
        LCD_ActiveRegion = 1;
        HAL_DSI_Refresh(hdsi);
        refr_qry = false;
    }
}

void HAL_DSI_EndOfRefreshCallback(DSI_HandleTypeDef *hdsi)
{

```

```

if(LCD_ActiveRegion < ZONES )
{
    /* Disable DSI Wrapper */
    __HAL_DSI_WRAPPER_DISABLE(hdsi);
    /* Update LTDC configuration */
    LTDC_LAYER(&hltdc_discovery, 0)->CFBAR = LAYER0_ADDRESS + LCD_ActiveRegion * HACT *
        2;
    __HAL_LTDC_RELOAD_CONFIG(&hltdc_discovery);
    __HAL_DSI_WRAPPER_ENABLE(hdsi);

    LCD_SetUpdateRegion(LCD_ActiveRegion++);
    /* Refresh the right part of the display */
    HAL_DSI_Refresh(hdsi);

}
else
{
    __HAL_DSI_WRAPPER_DISABLE(&hdsi_discovery);
    LTDC_LAYER(&hltdc_discovery, 0)->CFBAR = LAYER0_ADDRESS;

    __HAL_LTDC_RELOAD_CONFIG(&hltdc_discovery);
    __HAL_DSI_WRAPPER_ENABLE(&hdsi_discovery);

    LCD_SetUpdateRegion(0);
    lv_disp_flush_ready(disp);
}
}

#endif

/**
 * @brief Configure the DMA controller according to the Stream parameters
 *        defined in main.h file
 * @note This function is used to :
 *       -1- Enable DMA2 clock
 *       -2- Select the DMA functional Parameters
 *       -3- Select the DMA instance to be used for the transfer
 *       -4- Select Callbacks functions called after Transfer complete and
 *           Transfer error interrupt detection
 *       -5- Initialize the DMA stream
 *       -6- Configure NVIC for DMA transfer complete/error interrupts
 * @param None
 * @retval None
 */
static void DMA_Config(void)
{
/*##-1- Enable DMA2 clock #####*/
__HAL_RCC_DMA2_CLK_ENABLE();

/*##-2- Select the DMA functional Parameters ####*/
DmaHandle.Init.Channel = DMA_CHANNEL; /* DMA_CHANNEL_2 */
DmaHandle.Init.Direction = DMA_MEMORY_TO_MEMORY; /* M2M transfer mode */
DmaHandle.InitPeriphInc = DMA_PINC_ENABLE; /* Peripheral increment mode Enable */
DmaHandle.InitMemInc = DMA_MINC_ENABLE; /* Memory increment mode Enable */
#if LV_COLOR_DEPTH == 16
DmaHandle.Init.PeriphDataAlignment = DMA_PDATAALIGN_HALFWORD; /* Peripheral data alignment
    : 16bit */
DmaHandle.Init.MemDataAlignment = DMA_MDATAALIGN_HALFWORD; /* memory data alignment :
    16bit */
#else
DmaHandle.Init.PeriphDataAlignment = DMA_PDATAALIGN_WORD; /* Peripheral data alignment :
    16bit */
DmaHandle.Init.MemDataAlignment = DMA_MDATAALIGN_WORD; /* memory data alignment : 16bit */

```

```

#endif
DmaHandle.Init.Mode = DMA_NORMAL; /* Normal DMA mode */
DmaHandle.Init.Priority = DMA_PRIORITY_HIGH; /* priority level : high */
DmaHandle.Init.FIFOMode = DMA_FIFOMODE_ENABLE; /* FIFO mode enabled */
DmaHandle.Init.FIFOThreshold = DMA_FIFO_THRESHOLD_1QUARTERFULL; /* FIFO threshold: 1/4
    full */

DmaHandle.Init.MemBurst = DMA_MBURST_SINGLE; /* Memory burst */
DmaHandle.InitPeriphBurst = DMA_PBURST_SINGLE; /* Peripheral burst */

/*##-3- Select the DMA instance to be used for the transfer : DMA2_Stream2 */
DmaHandle.Instance = DMA_STREAM;

/*##-4- Initialize the DMA stream #####
if(HAL_DMA_Init(&DmaHandle) != HAL_OK)
{
    while(1);
}

/*##-5- Select Callbacks functions called after Transfer complete and Transfer error */
HAL_DMA_RegisterCallback(&DmaHandle, HAL_DMA_XFER_CPLT_CB_ID, DMA_TransferComplete);
HAL_DMA_RegisterCallback(&DmaHandle, HAL_DMA_XFER_ERROR_CB_ID, DMA_TransferError);

/*##-6- Configure NVIC for DMA transfer complete/error interrupts #####
HAL_NVIC_SetPriority(DMA_STREAM_IRQ, 0, 0);
HAL_NVIC_EnableIRQ(DMA_STREAM_IRQ);
}

/***
 * @brief DMA conversion complete callback
 * @note This function is executed when the transfer complete interrupt
 *       is generated
 * @retval None
 */
static void DMA_TransferComplete(DMA_HandleTypeDef *han)
{
    y_flush_act++;

    if(y_flush_act > y2_flush) {
#if TFT_NO_TEARING
        if(lv_disp_flush_is_last(disp)) refr_qry = true;
        else lv_disp_flush_ready(disp);
#else
        if(lv_disp_flush_is_last(disp)) HAL_DSI_Refresh(&hdmi_discovery);

        lv_disp_flush_ready(disp);
#endif
    } else {
        buf_to_flush += (x2_flush - x1_flush + 1) * 2;
        /*##-7- Start the DMA transfer using the interrupt mode #####
        /* Configure the source, destination and buffer size DMA fields and Start DMA Stream
           transfer */
        /* Enable All the DMA interrupts */
        if(HAL_DMA_Start_IT(han,(uint32_t)buf_to_flush, (uint32_t)&my_fb[y_flush_act *
                      TFT_HOR_RES + x1_flush],
                           (x2_flush - x1_flush + 1)) != HAL_OK)
        {
            while(1); /*Halt on error*/
        }
    }
}

/***
 * @brief DMA conversion error callback

```

```

 * @note This function is executed when the transfer error interrupt
 *       is generated during DMA transfer
 * @retval None
 */
static void DMA_TransferError(DMA_HandleTypeDef *han)
{
    while(1);
}

/***
 * @brief This function handles DMA Stream interrupt request.
 * @param None
 * @retval None
 */
void DMA_STREAM_IRQHANDLER(void)
{
    /* Check the interrupt and clear flag */
    HAL_DMA_IRQHandler(&DmaHandle);
}

/***
 * @brief Initialize the BSP LCD Msp.
 * Do not DMA2D is initialized by LVGL
 */
void BSP_LCD_MspInit(void)
{
    /** @brief Enable the LTDC clock */
    __HAL_RCC_LTDC_CLK_ENABLE();

    /** @brief Toggle Sw reset of LTDC IP */
    __HAL_RCC_LTDC_FORCE_RESET();
    __HAL_RCC_LTDC_RELEASE_RESET();

    /** @brief Enable DSI Host and wrapper clocks */
    __HAL_RCC_DSI_CLK_ENABLE();

    /** @brief Soft Reset the DSI Host and wrapper */
    __HAL_RCC_DSI_FORCE_RESET();
    __HAL_RCC_DSI_RELEASE_RESET();

    /** @brief NVIC configuration for LTDC interrupt that is now enabled */
    HAL_NVIC_SetPriority(LTDC_IRQn, 3, 0);
    HAL_NVIC_EnableIRQ(LTDC_IRQn);

    /** @brief NVIC configuration for DSI interrupt that is now enabled */
    HAL_NVIC_SetPriority(DSI_IRQn, 3, 0);
    HAL_NVIC_EnableIRQ(DSI_IRQn);
}

void DSI_IRQHandler(void){
    HAL_DSI_IRQHandler(&hdsi_Discovery);
}

```

Fragmento 11: Fichero Touchpad.h con las estructuras de mensajes

```

 /**
 * @file indev.h
 *
 */
#ifndef INDEV_H
#define INDEV_H

```

```

/*****
 *      INCLUDES
 *****/
#include <stdbool.h>
#include <stdint.h>

/*****
 *      DEFINES
 *****/
/*****
 *      TYPEDEFS
 *****/
/*****
 * GLOBAL PROTOTYPES
 *****/
void touchpad_init(void);

/*****
 *      MACROS
 *****/
#endif

```

Fragmento 12: Fichero Touchpad.c

```

/**
 * @file indev.c
 *
 */

/*****
 *      INCLUDES
 *****/
#include "tft.h"
#include "lvgl.h"

#include "stm32f7xx.h"
#include "stm32f769i_discovery.h"
#include "stm32f769i_discovery_ts.h"

/*****
 *      DEFINES
 *****/
/*****
 *      TYPEDEFS
 *****/
/*****
 * STATIC PROTOTYPES
 *****/
static void touchpad_read_cb(lv_indev_t * indev, lv_indev_data_t *data);

/*****
 * STATIC VARIABLES
 *****/
static TS_StateTypeDef TS_State;

/*****
 * MACROS
 *****/

```

```
******/  
  
*****  
* GLOBAL FUNCTIONS  
*****/  
  
/**  
 * Initialize your input devices here  
 */  
void touchpad_init(void)  
{  
    BSP_TS_Init(TFT_HOR_RES, TFT_VER_RES);  
    lv_indev_t * indev = lv_indev_create();  
    lv_indev_set_type(indev, LV_INDEV_TYPE_POINTER);  
    lv_indev_set_read_cb(indev, touchpad_read_cb);  
}  
  
*****  
* STATIC FUNCTIONS  
*****/  
  
static void touchpad_read_cb(lv_indev_t * indev, lv_indev_data_t *data)  
{  
    static int16_t last_x = 0;  
    static int16_t last_y = 0;  
    BSP_TS_GetState(&TS_State);  
    if(TS_State.touchDetected != 0) {  
        data->point.x = TS_State.touchX[0];  
        data->point.y = TS_State.touchY[0];  
        last_x = data->point.x;  
        last_y = data->point.y;  
        data->state = LV_INDEV_STATE_PR;  
    } else {  
        data->point.x = last_x;  
        data->point.y = last_y;  
        data->state = LV_INDEV_STATE_REL;  
    }  
}
```
