Sistemas Digitales II

Universidad Politécnica de Madrid

Departamento de Ingeniería Electrónica

E.T.S.I. de Telecomunicación

Memoria del proyecto desarrollado en Sistemas Digitales II (SDII)

Curso 2014/2015

*“Analizador de espectros en entorno de desarrollo con ANSI C”*

Autores:

Amadeo de Gracia Herranz

Daniel Montesano Martinez

# Código de la pareja: JT-2

ÍNDICE GENERAL

1 Código de la pareja: JT-2 1

2 Introducción 2

2.1 Mejoras 2

2.1.1 Elección de modo de funcionamiento mediante el teclado matricial. 2

3 Diagrama de subsistemas 3

3.1 Hardware 3

3.1.1 Adaptación y Filtrado 3

3.1.2 Amplificador 3

3.1.3 Filtrado 3

3.1.4 Generador rampa 3

3.2 Software 4

3.2.1 Configuración 4

3.2.2 Selección del modo de funcionamiento 4

3.2.3 Configuración del modo de funcionamiento 4

3.2.4 Proceso principal 4

3.2.5 Atención a la interrupción periódica 4

4 Descripción del subsistema Hardware 5

4.1 Adaptación y filtrado 5

4.2 Amplificador 5

4.3 Filtrado 6

4.4 Generador de rampa 6

5 Descripción del subsistema Software 8

5.1 Proceso del programa principal 8

5.1.1 Función inicialización del sistema 8

5.1.2 Función de selección de modo de funcionamiento 9

5.1.3 Función de configuración del modo de funcionamiento 9

5.1.4 Subrutina Representación de frecuencia 9

5.1.5 Subrutina Pulso de reset 9

5.2 Proceso de la interrupción periódica para muestreo 9

6 Descripción de las mejoras 10

6.1 Selección de modo de funcionamiento 10

7 Principales problemas encontrados 12

8 Anexos 13

# Introducción

El objetivo del proyecto es la realización en ANSI C de un analizador de espectros basado en MCF5272 y un osciloscopio para la visualización. Personalmente hemos tratado el proyecto como una manera de aprender el lenguaje C lo más estandarizado posible para poder aplicarlo en otros proyectos futuros, permitiendo también portar dicho proyecto a otros procesadores.

La práctica se podría resumir como:

Procesado de señales en tiempo real.

## Mejoras

### Elección de modo de funcionamiento mediante el teclado matricial.

Al encender nuestro sistema este esperará a que el usuario introduzca el modo de funcionamiento mediante un teclado matricial.

# Diagrama de subsistemas

## Hardware



### Adaptación y Filtrado

Adaptamos la impedancia de la salida del reproductor a nuestro sistema haciendo que nuestro sistema tenga una impedancia de entrada de 33 Ω. Se elimina la posible continua colocando un filtro pasivo RC a una fc=50Hz.

### Amplificador

Debido a las diferencias de potencia de salida entre unos reproductores y otros colocamos un amplificador de ganancia variable con el fin de tener siempre 1V de amplitud a la entrada de nuestro ADC.

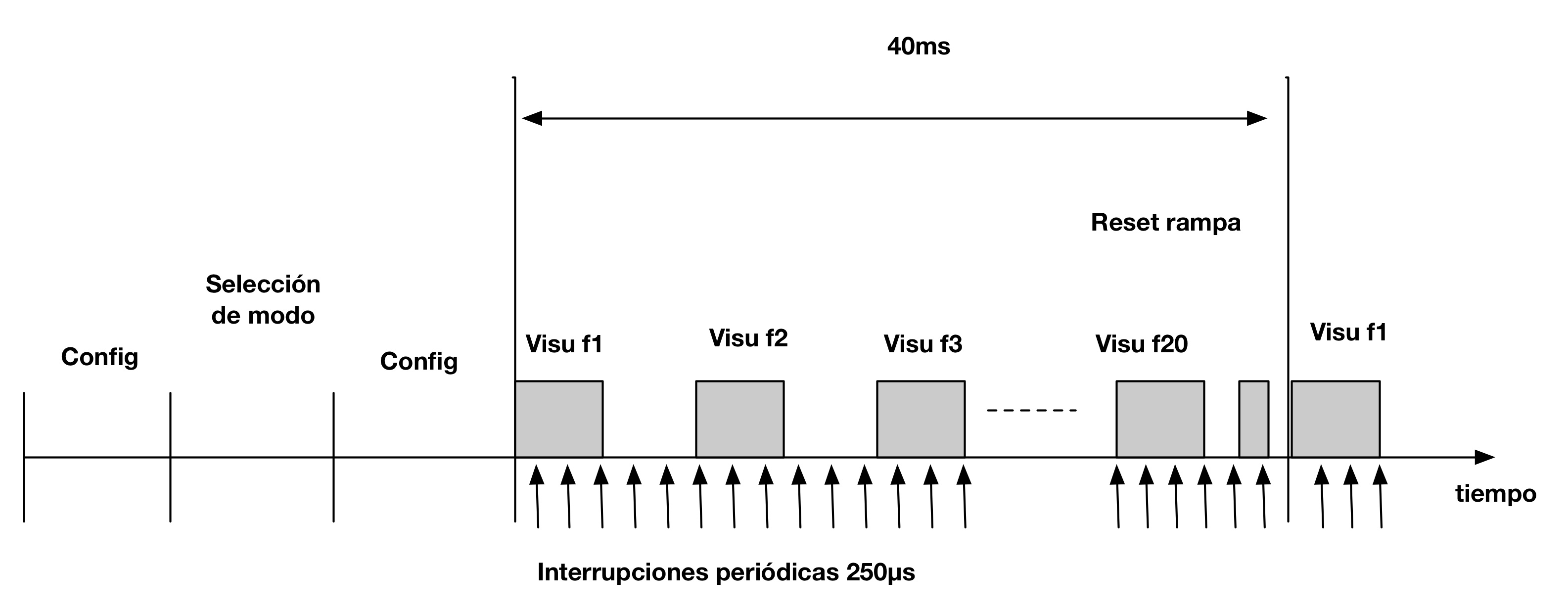
### Filtrado

Elimina las frecuencias por encima de las que nuestro ADC no sería capaz de muestrear correctamente, de esta manera evitamos el solapamiento durante el muestreo.

### Generador rampa

Genera una rampa que nos permite barrer el eje x del osciloscopio en modo xy.

## Software



### Configuración

Se configuran los parámetros necesarios para que el procesador pueda empezar a funcionar.

### Selección del modo de funcionamiento

Mediante el teclado matricial se escoge el modo de funcionamiento. Debemos tener en cuenta que la configuración de las interrupciones dependerá del modo de funcionamiento que el usuario ha indicado.

### Configuración del modo de funcionamiento

Configuración de los parámetros directamente dependientes del modo de funcionamiento.

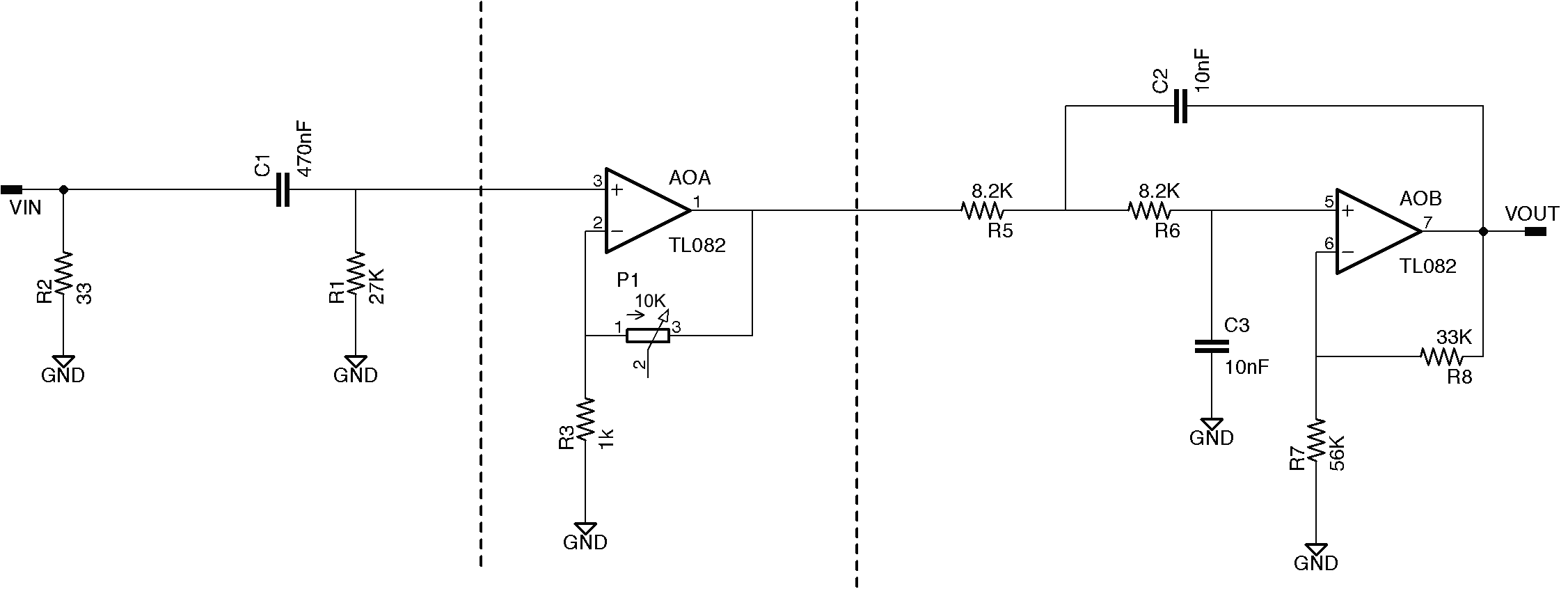
### Proceso principal

Se encargara de la visualización de los datos obtenidos. Cada 2ms se cambia el valor de salida del DAC y cada 40ms se emite un pulso de reset para reiniciar la rampa.

### Atención a la interrupción periódica

Se obtienen los datos del ADC cada 250µs, se llamara a las funciones necesarias para realizar el cálculo de la DFT almacenando los resultados en memoria.

# Descripción del subsistema Hardware



|  |  |  |
| --- | --- | --- |
| Adaptación y filtrado | Amplificador | Filtrado |

Para comprobar el correcto funcionamiento del subsistema de adaptación y filtrado, se realizaron diferentes pruebas. En primer lugar, se hizo una simulación por ordenador en el programa 5spice, comprobando así que la ganancia con los componentes elegidos era la correcta. Tras el montaje de la electrónica, probamos el sistema completo, introduciendo una señal de un reproductor a la entrada y ajustando el potenciómetro para tener 1 V de amplitud a la salida. Sin embargo, hubo problemas con la resistencia de adaptación, ya que al ser demasiado pequeña, la señal desaparecía. La solución más sencilla fue cambiar la resistencia por una un poco mayor.

Para la parte de generación de rampa hemos seguido un procedimiento similar: una simulación en ordenador y una posterior prueba introduciendo señales desde el microcontrolador.

## Adaptación y filtrado

Frecuencia de corte filtro paso alto pasivo 50Hz, ya que el sistema esta especificado para frecuencias superiores a esta. Esta se calcula como 1/2πRC damos un valor comercial a la C (470nF) y a partir de ahí hallamos la R (6.8kΩ).

Pese a que en el enunciado se recomendaba colocar una R de carga de 10Ω hemos acabado colocando una de 33Ω. Cuando realizábamos medidas tal y como se recomendaba en el enunciado, sin la R de la entrada y con una señal de entrada sinusoidal, podíamos obtener en la salida total del sistema 1V de amplitud. Por otro lado intentamos realizar la medida colocando a la entrada un reproductor de música a máxima potencia esta vez colocando a la entrada la resistencia recomendada de 10Ω. Tras medir a la salida observábamos que no podíamos obtener una amplitud de 1V por lo que decidimos aumentar la resistencia de hasta 33Ω.

## Amplificador

Necesitamos que tenga una ganancia variable para que la entrada al ADC tenga 1V de amplitud independientemente de la amplitud que ofrezca el reproductor a la entrada del sistema. Suponemos una entrada de entre 50mV y 200mV por lo que la ganancia total del sistema deberá variar entre 5 y 20. Conocemos la ganancia del filtro posterior (1.585) por lo que la ganancia de nuestro amplificador debe poder variar entre 3.15 y 12.6. Decidimos colocar una R3 fija de 1kΩ y una variable de 10kΩ lo que nos permite ir desde ganancia 1 hasta 11 cubriendo razonablemente el rango de ganancias que habíamos planteado con una resolución adecuada.

## Filtrado

Factor de calidad: 1/√2

Frecuencia de corte: 2kHz

Condensadores: 10nF

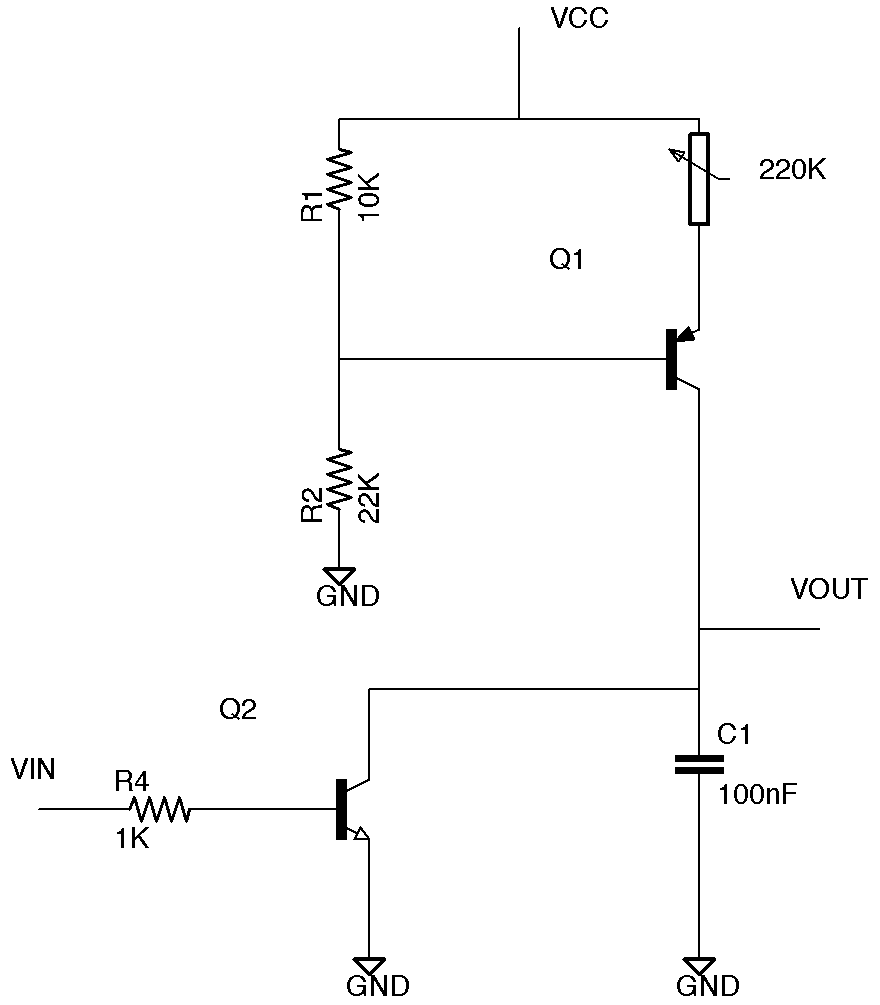
Resistencias: 8.2kΩ

R4: 56kΩ

R5: 33kΩ

G: 1.585

## Generador de rampa



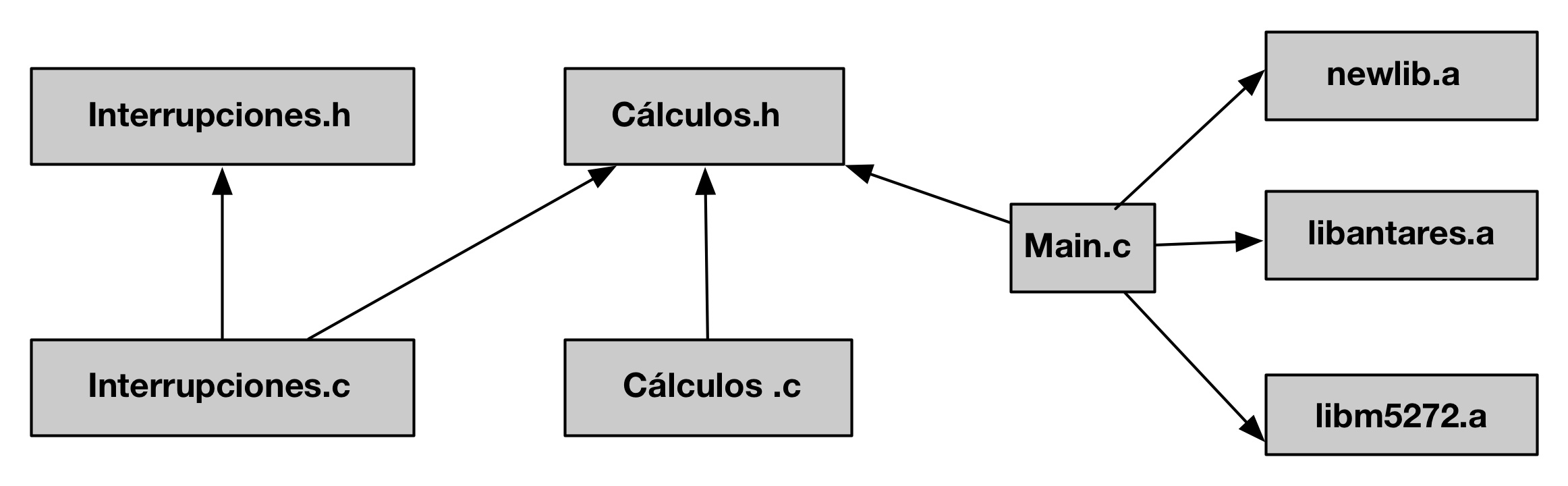
Conocemos que la duración de la rampa debe ser 40ms y el valor máximo de esta 2V.

Para los cálculos suponemos una Rvariable fijada en 200kΩ, valor del potenciómetro recomendado.

Ahora ya tenemos el orden de magnitud del condensador pero un condensador de 86nF no es comercial por lo que decidimos colocar uno de 100nF y rehacemos los cálculos en sentido inverso.

Pese a que esta resistencia no sea comercial es viable obtenerla a partir del uso de un potenciómetro de 220kΩ que es un valor comercial.

# Descripción del subsistema Software

Nuestro sistema software es un sistema modular. El código se agrupa por funcionalidades. Esto nos permite manejar de manera mas cómoda la complejidad del sistema. Cada modulo se abstrae de los demás, y solo necesita saber su definición, no su implementación. La principal ventaja de este sistema es que permite compilar cada modulo por separado, recompilando durante el proyecto solo los que sean necesarios. Nuestra estructura de ficheros y su relación es la siguiente:

## Proceso del programa principal

Inicialización del sistema

Selección del modo de funcionamiento esperando al teclado matricial

Configuración de lo necesario para el modo de funcionamiento escogido

Inicio de bucle principal

Deshabilita interrupciones

Representa frecuencia 1

Habilita interrupciones

Espera 2 ms

Deshabilita interrupciones

Representa frecuencia 2

Habilita interrupciones

Espera 2 ms

Deshabilita interrupciones

Representa frecuencia 3

*…*

Representa frecuencia 20

Habilita interrupciones

Pulso de reset.

*Fin del bucle*

### Función inicialización del sistema

CRT0: Inicialización de la ram.

Se encarga de inicializar lo necesario para iniciar el sistema. En este caso, solo se necesita inicializar la ram. No necesita parámetros de entrada ni devuelve parámetros de salida.

CRT1: BSS a cero

Nos aseguramos de que el BSS se pone a cero, ya que el compilador asume que este esta a cero.

Inicializacion de interfaz de usuario:

Se inicializa el LCD y el teclado matricial para que el usuario pueda interactuar con el sistema, asi como el DAC y el ADC.

### Función de selección de modo de funcionamiento

*Bucle*

Si se detecta una tecla correspondiente a un modo de funcionamiento, break.

Else, continue.

Se espera a que el usuario introduzca por el teclado matricial el modo de funcionamiento. No tiene variables de entrada y tiene la variable de salida que es el modo de funcionamiento.

### Función de configuración del modo de funcionamiento

Configuración de interrupciones periódicas.

### Subrutina Representación de frecuencia

Lee los datos almacenados en la ram.

Manda los datos al DAC.

Se encarga de la visualización de los resultados. Recoge los datos almacenados en memoria por las rutinas de cálculo y los transmite al DAC para su conversión a un voltaje y su posterior representación en un osciloscopio. Los parámetros de entrada son los datos a representar y no tiene variables de salida. Para evitar colisiones entre el DAC y el ADC, se deshabilitan las interrupciones mientras se escribe en el DAC.

### Subrutina Pulso de reset

Pin de salida a 1

Espera

Pin de salida a 0

Se encarga de activar durante el tiempo necesario y desactivar un pin de salida para descargar el condensador del generador de rampa.

## Proceso de la interrupción periódica para muestreo

Lee datos del ADC

Cálculo acumulativo de senos y cosenos de cada frecuencia

Si se han procesado 80 muestras

*Cálculo del módulo de cada frecuencia*

Else : continue

Fin de la rutina a la interrupción periodica

# Descripción de las mejoras

## Entorno de desarrollo estándar

Consideramos desarrollar este proyecto en ANSI C como una gran mejora. Esto, a su vez, nos ha permitido una serie de ventajas que se describirán a continuación.

Como base de todo, tenemos un buen entorno de desarrollo con un buen compilador, tanto en nativo como en cruzado. Es transparente, por lo que siempre sabemos que es lo que esta pasando por debajo. Se puede ver cada estado del proceso de compilación, desde la creación de los objetos hasta el linkeado de estos. También nos permite seguir una buena organización modular, importando las definiciones desde los ficheros .h y haciendo las implementaciones en los ficheros .c. Esto se traduce en poder tener una buena colección de librerías, como es por ejemplo newlib, lo que nos proporciona la mayoría de funciones de ANSI C.

Este entorno de desarrollo también nos ha permitido iniciar el programa con crt0 y crt1. Esto nos asegura que el BSS será inicializado a 0, tal y como debe ser, lo cual nos ahorra muchos problemas a la hora de la ejecución del programa, problemas ajenos al desarrollo del proyecto. Además nos asegura tener una salida ordenada. Cuando el programa se termina, la salida se produce de forma ordenada, evitando que el sistema se quede activo haciendo cosas no deseadas.

Por otro lado, tenemos un buen build system basado en makefiles. Esto nos permite compilar todos los cambios con un solo comando. Esto lleva implícito la descripción de la relación de todos los ficheros, de tal manera que cualquier persona ajena a este proyecto podría entender como esta estructurado.

Por ultimo, este entorno permite la ampliación de las herramientas (lo cual no hemos podido llegar a hacer por falta de tiempo) como por ejemplo un frontend común, como puede ser eclipse, o cargar directamente desde el propio ordenador desde una consola de puerto serie.

## Programación portable

Nuestra practica surgió a partir de la decisión de hacer un código lo mas portable posible. Para ello, hemos programado en el C mas estándar posible, permitiéndonos así poder usar este código tanto en el m5272 como en muchos otros procesadores. Esto no fue sencillo, ya que muchas de las funciones de la librería que se nos proporciona en el laboratorio son funciones propias y no están definidas para ningún otro sistema. Esto nos llevo a tener que adaptar dichas funciones a las estándar de C. Por ejemplo, ahora se puede usar la función nanosleep para especificar un retardo. También existe la función putchar. Esto implicaba unir varias librerías: la librería proporcionada en el laboratorio, nuestra librería propia creada para poder simular y newlib del proyecto Antares

## Simulación en nativo

La mejora descrita anteriormente nos permite compilar y ejecutar en cualquier otro ordenador. Esto nos ha permitido probar el programa y la mayoría de las mejoras en nuestros ordenadores, y no tener que ir al laboratorio para comprobar el código escrito. Hemos desarrollado gran parte del código en nuestra casa, y yendo solo al laboratorio para probarlo.

Para llevar a cabo esta mejora, se ha creado una librería, en la cual se definen las mismas funciones que la librería que se proporciona en el laboratorio pero se implementan de manera diferente. Esto es debido a que algunas funciones no tienen mucho sentido en un ordenador. Por ejemplo, escribir en un LCD será escribir en consola y la interrupción periódica no es una interrupción del procesador como tal, si no que es un thread aparte, que llama a una función cada tiempo determinado. Es importante destacar que la interrupción periódica lee un fichero de texto con los valores que sacaría un ADC como el de la placa de desarrollo que hay en los laboratorios. Para mostrar los resultados, estos se escriben en un fichero de texto, poniendo el valor que se pondría al DAC. Esto implica que al adquisición de datos y el procesamiento de estos son dos procesos completamente separados, lo cual nos permitiría la adquisición de datos en un momento determinado (también pueden ser creados artificialmente, como por ejemplo, sinusoides perfectas o sumas de estas, para lo cual hemos creado también una pequeña función en C que genera la suma escalada como el ADC de las sinusoides de las frecuencias que se le pasan como argumento) y el procesado de estos mas tarde. También se pueden volver a procesar exactamente los mismos datos para comprobar que el sistema se comporta siempre igual para una misma entrada.

Hay que destacar que esto se puede hacer debido a que tenemos un buen sistema de manejo de complejidad, tal y como se ha dicho antes, pudiendo así hacer un sistema modular, separando las implementaciones de las definiciones.

## Compilación cruzada

Gracias a tener un compilador cruzado que podemos ejecutar en nuestras máquinas somos capaces de detectar errores de compilación desde casa sin necesidad de invertir tiempo en el laboratorio. Esto implica depender lo mínimo posible de un entorno de desarrollo desconocido y poco usado, y que causa una gran cantidad de problemas ajenos al proyecto. La idea principal era compilar en nuestros ordenadores para mas tarde transmitir los datos mediante puerto serie directamente al m5272. Pero debido a la falta de tiempo y la falta de documentación del puerto serie de Antares no se pudo llevar a cabo, y acabamos cargando el binario a través del EdColdFire.

## Uso de Makefile

La compilación de todo este programa línea a línea resultaría compleja y tediosa tarea que se ve simplificada gracias al Makefile y la herramienta make ya que toda la compilación del proyecto se ve reducida a ejecutar en la consola un sencillo “make all”. Típicamente en la programación en C se escribe un archivo Makefile en el cual se definen las dependencias entre los ficheros que forman el programa así como las distintas reglas o normas que se deben seguir a la hora de su compilación.

Nosotros hemos creado este fichero junto con una serie de Makeconf asociados, con el fin de que con cambiar un simple “flag” en uno de los ficheros sea posible compilar en nativo o para el m5272. Por otro lado los Makeconf nos sirven también para poder variar en función de si vamos a compilar para Linux (Ubuntu) o para MacOS ya que el lugar en el que están ubicadas algunas de las librerías varían en función del sistema operativo.

## Manejo de un LCD como salida de la interfaz de usuario

Usamos el LCD para mostrar información al usuario. Son ordenes sencillas, como permitirle seleccionar el modo en el que funcionara el sistema, ya que el LCD solo tiene unos pocos caracteres, y queremos que la orden se muestre completa y se muestre todo el rato (lo cual no ocurriría si se desplazasen caracteres para mostrar una orden mas larga). La gran ventaja que tiene usar un LCD frente a la consola del ordenador es que mandar comandos por el puerto serie es un proceso complicado y bastante lento. Si la frecuencia escritura de ordenes en pantalla fuera elevada, podría hacer que el sistema dejase de ser a tiempo real.

## Manejo de un teclado matricial como entrada de la interfaz de usuario

El teclado matricial nos permite interactuar con el sistema. La otra forma posible de interactuar con el sistema es la consola del puerto serie y a base de comandos, lo cual no es sencillo de usar y además es un proceso lento. Al igual que con el LCD, al usar el teclado matricial para meter ordenes predefinidas conseguimos ahorrar tiempo y hacer que el sistema pueda ser un sistema a tiempo real.

## Selección de modo de funcionamiento

Cuando nuestro programa se inicia esperara a que el usuario seleccione el modo de funcionamiento mediante el teclado matricial. Dependiendo de si estamos compilando en cruzado o simulando en nativo, los diferentes modos llamaran a funciones diferentes, pero en definitiva el cada modo hará la misma funcionalidad. Los diferentes modos implementados ahora mismo son:

Modo principal:

Al pulsar 1, se cargara el modo principal, el cual ya ha sido descrito en esta memoria (lee datos del ADC, calcula la DFT, saca valores por DAC)

Modo testDAC:

Sirve para probar el DAC

Modo testADC\_DAC:

Sirve para probar el ADC, asumiendo que el DAC funciona antes.

## Control de versiones

Pese a no ser algo intrínseco al proyecto, consideramos el haber usado GitHub un hecho a destacar, ya que permite tener un control de la evolución del proyecto, así como el libre acceso para cualquier persona que lo necesite.

# Valor añadido

Creemos que nuestro proyecto puede tener un gran valor añadido para el laboratorio. Futuras practicas que usasen como base nuestro proyecto podrían tener muchas ventajas para los alumnos.

El laboratorio se podría hacer en gran medida en casa. Los alumnos irían solo al laboratorio para probar el código que han hecho en casa. Esto haría que el laboratorio no estuviera tan saturado, y que no seria estrictamente necesario hacer toda la practica en el laboratorio.

Por otro lado, permitiría que los alumnos usasen un entorno de desarrollo que puede ser usado perfectamente fuera del laboratorio y en cualquier otro proyecto. Esto permite que se familiaricen con herramientas como make o los diferentes compiladores. También herramientas de debug como el debugger de gcc o las binutils. Además, estas herramientas pueden ser actualizadas fácilmente.

En segundo lugar, permite que los alumnos se acostumbren a escribir en ANSI C, lo cual es una clara ventaja mirando un futuro próximo en el que necesite programar cualquier otra cosa en C. ANSI C es un estándar en la industria. Tambien, simplifica el laboratorio, ya que reduce la complejidad accidental frente a la complejidad intrínseca del proyecto.

# Principales problemas encontrados

### Hardware

Resistencia de entrada al sistema, especificado en el apartado 3.1

### Falta de documentación

Al ser un procesador obsoleto, encontrar documentación actualizada sobre él no es nada sencillo. El mayor problema que encontramos fue cuando, tras cargar nuestro programa en la placa de entrenamiento y ejecutarlo, el programa acaba parándose. Esto se debía a que el compilador había generado una instrucción que el procesador no entendía. Acabamos solucionando el problema utilizando una versión mas antigua del compilador.

### Herramientas de desarrollo

Ya que el laboratorio solo proporciona un entorno de desarrollo que no satisfacía nuestras necesidades, nos vimos obligados a buscar otro entorno de desarrollo y adaptarlo, lo cual nos llevo gran cantidad de tiempo.

### Poco tiempo para prueba

No conseguimos la primera versión del compilador cruzado hasta finales de abril. Como se ha explicado en el apartado anterior, esta primera versión generaba instrucciones desconocidas por el procesador. El tiempo invertido en solventar este problema nos ha dejado poco tiempo para realizar mejoras sobre la aplicación. Si hubiéramos tenido mas tiempo se podrían haber implementado algunas mejoras de las que se ha hablado, como el debugger, un buen frontend o comunicación serie desde otros ordenadores.

### Cargado en la placa

La carga del programa principal en la placa se realiza por medio de un puerto serie. Intentamos realizar esta carga desde el puerto serie de nuestros ordenadores, pero no fuimos capaces de conseguirlo. Consideramos que este tema podría haberse conseguido con algo más de tiempo. La solución por la que hemos optado ha sido cargarlo desde el entorno de desarrollo en los puestos de laboratorio. Realizar la carga desde el entorno desencadenó un nuevo problema, necesitábamos tener un archivo con el desensamblado además del archivo que se carga en la placa. Este fichero innecesario para el la carga tardaba mucho tiempo en generarse, lo cual hacia que las pruebas en la placa fueran lentas y tediosas.

# Anexos

Todo el código referente al proyecto se puede encontrar en: [github.com/amadeodeg/SDG2](https://github.com/amadeodeg/SDG2)