401

Microcontroladores II

Trabajo Práctico Final

Implementación de Filtro FIR para Audio en STM32

Autor	**Legajo**	**Docente**	**Fecha**
Juan Bautista Olivera	152254241	Ing. Maximiliano Vega	26/06/2025

Índice

Introducción

El presente trabajo práctico tiene como finalidad el diseño e implementación de un sistema de procesamiento digital de señales acústicas utilizando un microcontrolador STM32F401RE, en el marco del curso de “Microcontroladores II” dictado en la Universidad Católica Argentina. El objetivo central es desarrollar un filtro digital de tipo FIR capaz de atenuar las componentes agudas de una señal de voz humana, adquirida mediante un micrófono EMC-50, y reproducir la señal resultante en un parlante a través de un sistema de amplificación. El sistema completo deberá operar en tiempo real, ejecutando el filtrado en la propia unidad de procesamiento embebida.

Este trabajo integra conocimientos de diseño electrónico, programación en sistemas embebidos y procesamiento digital de señales, y busca consolidar su aplicación mediante un desarrollo funcional y experimental que permita validar el comportamiento del sistema.

Objetivos

El objetivo general del proyecto es desarrollar un sistema embebido que permita realizar el filtrado digital en tiempo real de señales de voz humana, atenuando sus componentes agudas a través de la implementación de un filtro FIR en la plataforma NUCLEO-F401RE. Entre los objetivos específicos se destacan:

- Seleccionar e integrar los módulos de hardware requeridos, garantizando la compatibilidad entre niveles de señal y protocolos.
- Diseñar el filtro FIR aplicando técnicas de análisis y simulación mediante scripts en Python, con base en criterios acústicos apropiados para el procesamiento de voz.
- Implementar la lógica de adquisición, procesamiento y reproducción de señales en la plataforma STM32 utilizando sus periféricos internos.
- Proponer mejoras en la flexibilidad del sistema, incluyendo la posibilidad de modificar los parámetros del filtro mediante una interfaz configurable por el usuario.

Marco teórico

El procesamiento digital de señales (DSP) permite operar sobre señales discretas en el tiempo con el fin de modificarlas o extraer información útil. En esta práctica, se emplea un filtro digital de tipo FIR (Finite Impulse Response), el cual posee una estructura no recursiva que garantiza estabilidad incondicional. Estos filtros se definen por una secuencia finita de coeficientes multiplicativos, aplicados directamente a una ventana de muestras de la señal de entrada.

El diseño de un filtro FIR requiere establecer criterios de filtrado tales como la frecuencia de corte, el orden del filtro, el tipo de ventana utilizada, entre otros. En este trabajo, el diseño se realiza mediante simulaciones y pruebas en Python, herramienta que permite verificar la respuesta del filtro antes de su implementación en el entorno embebido.

La adquisición de la señal analógica proveniente del micrófono se realiza a través de un conversor analógico-digital (ADC) interno del microcontrolador STM32F401RE. Este ADC permite una resolución de hasta 12 bits y una frecuencia de muestreo programable. Según el teorema de muestreo de Nyquist, la frecuencia de muestreo debe ser, como mínimo, el doble de la mayor frecuencia presente en la señal para evitar el fenómeno de aliasing.

Para adecuar la señal del micrófono EMC-50 a los niveles requeridos por el ADC, se emplea un amplificador de audio LM386. Este circuito integrado permite amplificar señales de baja potencia con un bajo número de componentes externos y alta eficiencia. La salida del amplificador es luego acondicionada y digitalizada por el ADC, alimentando así el proceso de filtrado digital.

Finalmente, la señal procesada es convertida nuevamente al dominio analógico mediante el uso de una técnica de modulación por ancho de pulso (PWM), aprovechando los temporizadores del microcontrolador, y es enviada a un parlante de 4 Ω mediante un filtro pasa bajos.

Implementación

Preamplificador del Microfono

El siguiente apartado describe el diseño y justificación del sistema de acondicionamiento de señal proveniente de un micrófono, orientado a su posterior digitalización por medio del conversor analógico digital (ADC) de un microcontrolador STM32. Este sistema consta de dos etapas: una primera de amplificación, implementada con un amplificador de audio LM386, y una segunda de protección y adecuación de nivel, llevada a cabo mediante un optoacoplador del tipo 4N35. La simulación completa fue realizada en KiCad, tanto para verificar la topología de conexión como para observar el comportamiento de la señal.

Amplificador Principal

La etapa principal de amplificación está basada en el circuito integrado LM386, utilizando la configuración del datasheet para ofrecer una ganancia de 50 veces. Esta elección se justifica a partir de los requerimientos del sistema y de la simulación ya que se necesita amplificar señales de baja amplitud provenientes del micrófono a niveles compatibles con el ADC de la STM32, sin introducir distorsiones ni complicaciones en el diseño y mediante simulaciones se llegó a la conclusión de que una ganancia de 20 veces no es suficiente.

El LM386 está diseñado específicamente para aplicaciones de audio, operando con una sola fuente de alimentación de bajo voltaje (en este caso se utilizará 5V), con una alta ganancia interna y una baja cantidad de componentes externos. A diferencia de un amplificador operacional genérico no requiere una configuración diferencial y posee una impedancia de entrada adaptada a fuentes de baja potencia. Además, su ganancia puede ajustarse entre 20 y 200 mediante un capacitor y resistencias entre los pines 1 y 8, lo cual facilita la sintonización según el tipo de micrófono utilizado.

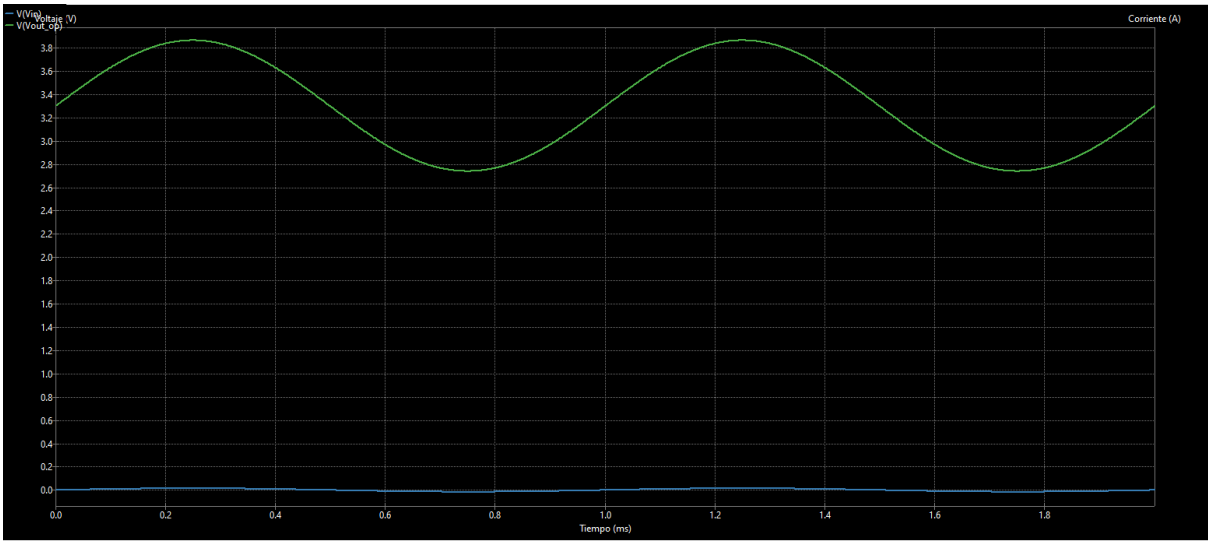


Figura 1: Simulación de KiCAD donde se puede ver la entrada de 10mVpp y la salida del LM386

En la simulación en KiCad (Figura 1), se observa cómo una señal de entrada de aproximadamente 10 mVpp se amplifica a una salida de 1 Vpp, cumpliendo con el rango de trabajo deseado. La topología del circuito se ilustra en el esquemático de la Figura 2.

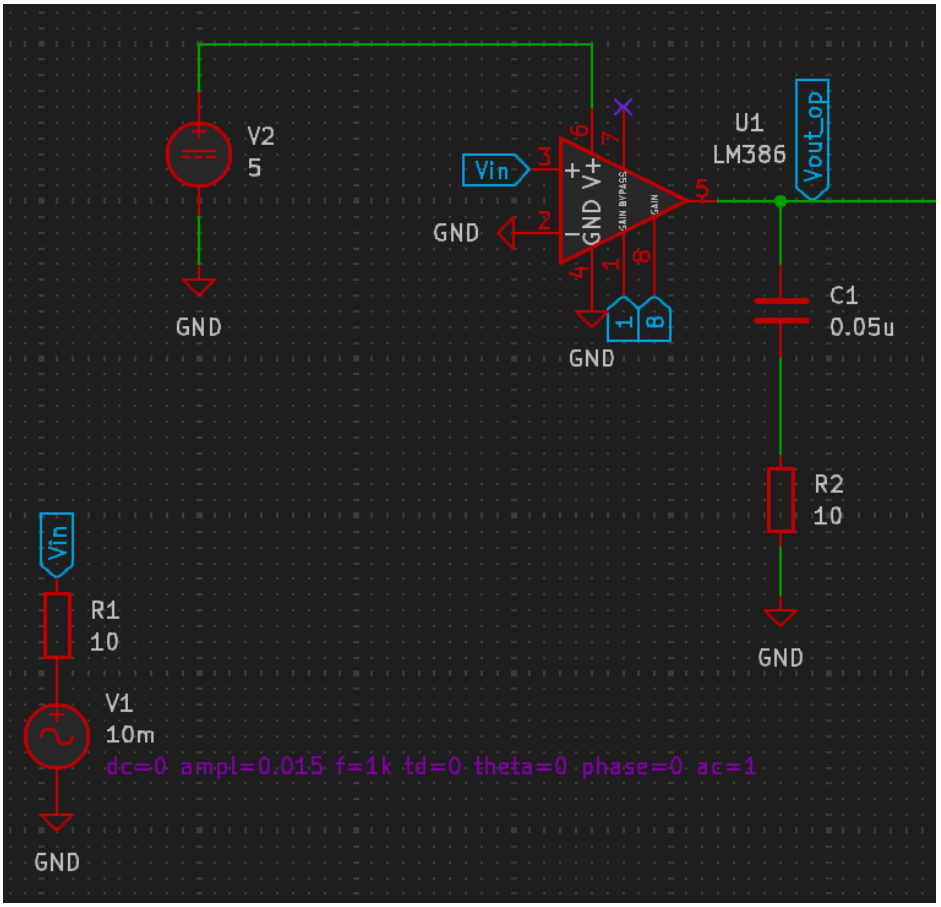


Figura 2: Esquemático del circuito utilizado con el LM386

Protección para el ADC

Una característica del LM386 es que, al ser alimentado con una fuente simple, la señal de salida contiene un componente de continua (offset) centrado en aproximadamente la mitad de la tensión de alimentación, en este caso 2.5V a 3V. Este comportamiento es usual en amplificadores con acoplamiento directo cuando no se usa un capacitor de salida para eliminar la componente continua.

Este offset representa un problema importante al momento de muestrear la señal con el ADC del mocrocontrolador ya que el mismo solo acepta tensiones comprendidas entre 0V y 3.3V. Si la señal amplificada presenta un valor pico que supera los 3.3V debido a su offset, existe el riesgo de quemarlo.

Con el objetivo de proteger al microcontrolador y acondicionar la señal para su digitalización, se optó por el uso de un optoacoplador tipo 4N35. Este componente proporciona aislamiento galvánico entre la etapa analógica y la etapa digital, desacoplando completamente el dominio de potenciales del amplificador y el ADC.

El funcionamiento del 4N35 se basa en un LED infrarrojo interno que, al conducir corriente, ilumina un fototransistor. Este último puede polarizarse para que su salida tenga un offset controlado, centrado idealmente en 1.65V, que es el punto medio del ADC de 3.3V.

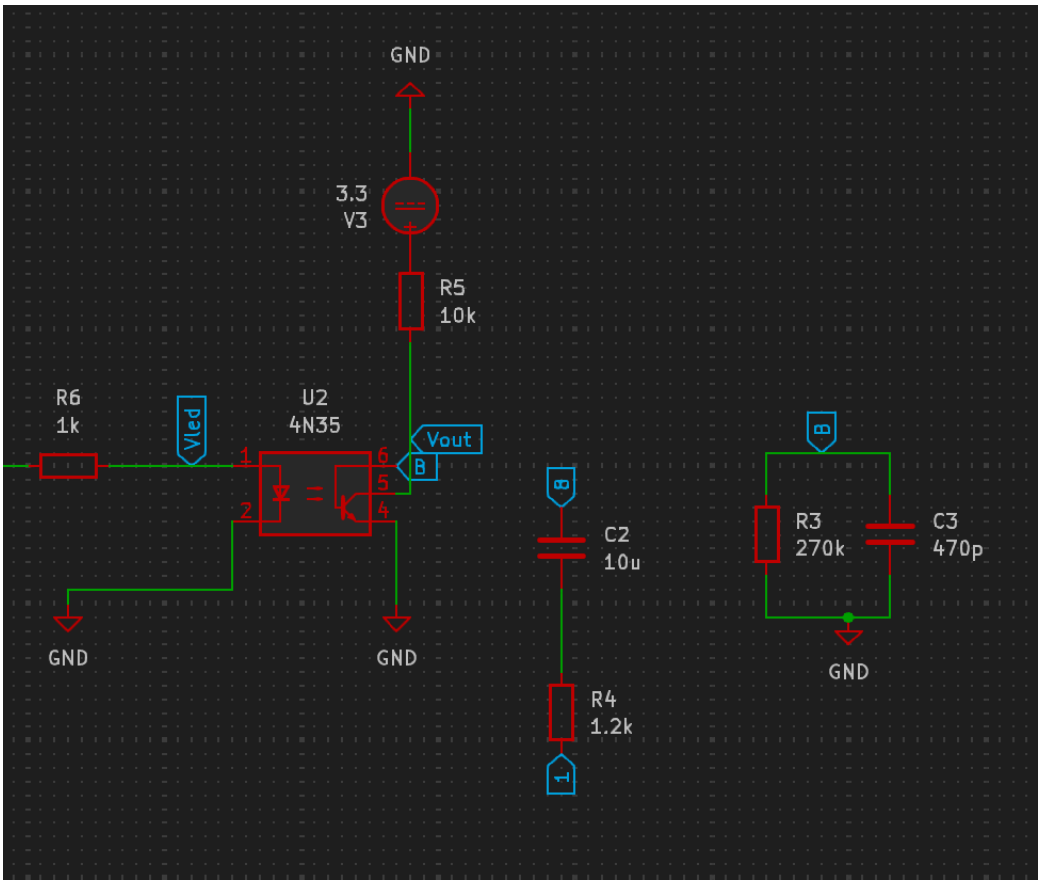


Figura 3: Esquemático del circuito utilizado con el 4N35

Adicionalmente, mediante el diseño del circuito colector-común del optoacoplador, se puede limitar la tensión máxima de salida a 3.3V, garantizando que ninguna componente transitoria o de ruido exceda dicho valor.

Según la hoja de datos oficial del 4N35 de Onsemi Semiconductor y Vishay, este tipo de protección resulta especialmente útil en entornos ruidosos o en sistemas sensibles como los basados en microcontroladores.

La simulación de esta etapa, visible en la Figura 4, permite corroborar que la señal de salida del optoacoplador no excede los 3.3V y está adecuadamente centrada. El esquemático correspondiente se presenta en la Figura 3.

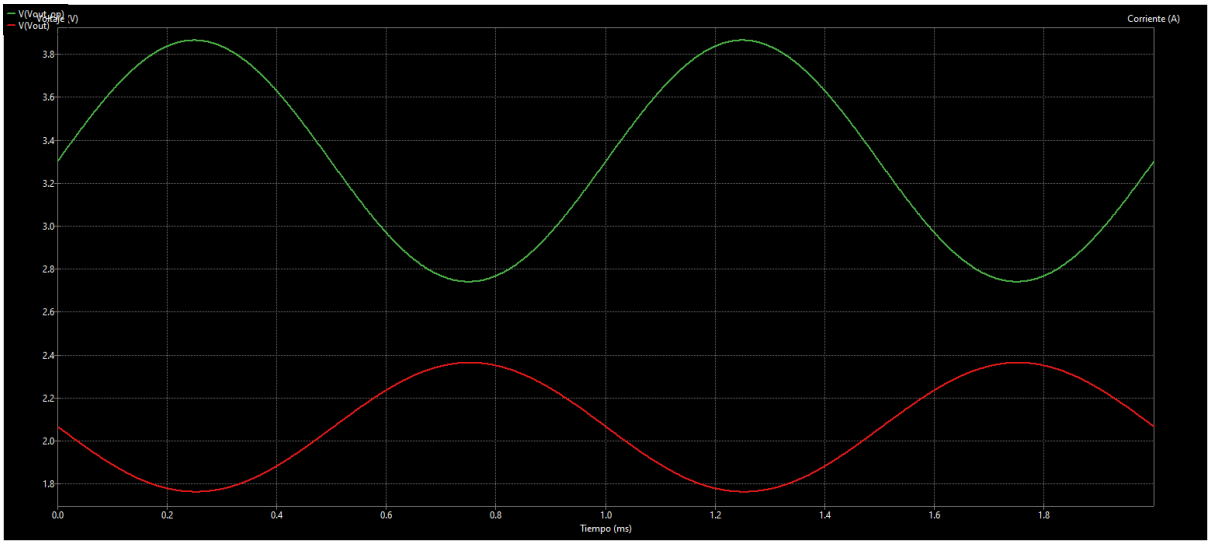


Figura 4: Simulación de KiCAD con la salida del LM386 y la salida del 4N35

Laboratorio

Durante el laboratorio, se hicieron un par de pruebas tanto del amplificador como del circuito de protección del ADC. Habiendo realizado esto y conversando con el docente, se llegó a las siguientes conclusiones:

1. La preocupación por parte del alumno por proteger el ADC es totalmente valida ya que, analizando el datasheet, el mismo no posee tanta tolerancia y el LM386 lo podria quemar.
2. La utilización de un optoacoplador para la protección de un microcontrolador no es la manera estandard de hacerlo. Generalmente se utilizan diodos puestos en alguna configuración.
3. El circuito soldado no funciono por alguna falla en la soldadura o porque se quemó el integrado realizando una prueba el dia antes.

Habiendo llegado a estas conclusiones, por razones de tiempo no se llega a realizar una placa y circuito nuevo, por lo que se optó por utilizar la placa del docente que posee un LM358.

Filtro FIR

Pre Implementación

Analizando el codigo para la STM32 proporcionado por el docente (Carpeta FIR), el sistema implementado utiliza una estrategia de muestreo periódica basada en interrupciones generadas por el Timer 3, y la posterior lectura manual del ADC desde el programa principal.

En el archivo my_lib.h se definen las macros de configuración del temporizador base:

```
#define CORE_CLK  16000000 // 16 MHz
```

```
#define Timer_Period ((uint32_t) 100) // 1 [ms]
#define _BasePeriod(x) (((x * (CORE_CLK/100000)) /
BASE_PRESCALER)-1)
```

Estas definiciones indican que el sistema posee un reloj principal de 16 MHz (CORE_CLK) y que se desea generar una interrupción cada 1 milisegundo (Timer_Period = 100, siendo que la unidad es de 10 µs). El macro _BasePeriod(x) convierte el período deseado a una cuenta de temporizador;

$$T = (100 \cdot \frac{16MHz}{100000}) - 1 = 15999$$

Esta cuenta es utilizada en la función de configuración del temporizador TIM3_Config() en my_lib.c, donde se inicializa el temporizador en modo de interrupción, generando la misma cada 1 milisegundo:

```
hbasetim.Init.Period = _BasePeriod(Timer_Period);
...
HAL_TIM_Base_Start_IT(&hbasetim);
```

La variable global sampling actúa como un contador decreciente que define la frecuencia efectiva de muestreo. En el archivo main.c, durante la inicialización del sistema, se establece:

```
sampling = ADC_Sampling_Period;
```

Donde ADC_Sampling_Period está definido en my_lib.h como:

```
#define ADC_Sampling_Period ((uint32_t) 10) // 10 [ms]
```

Esto indica que se tomará una muestra cada 10 interrupciones del temporizador TIM3 (es decir, cada 10 ms). La rutina de interrupción del temporizador (TIM3_IRQHandler) se encarga de disminuir el contador y reiniciarlo cuando llega a cero:

```
void TIM3_IRQHandler(void)
{
    if(sampling > 0) sampling--;
    else sampling = ADC_Sampling_Period;

    HAL_TIM_IRQHandler(&hbasetim);
}
```

En el bucle principal del programa (main.c), se verifica si la variable sampling ha llegado a cero para proceder a leer el valor del ADC:

```
while (1)
{
    if (sampling == 0)
    {
        new_sample = ADC_Read();
        filtered_value = FIR_Filter(new_sample);
    }
}
```

La función ADC_Read() realiza una lectura sincrónica del ADC y convierte el resultado a una tensión en voltios, utilizando la constante ADC_RES para la resolución:

```
float ADC_Read(void){
    HAL_ADC_Start(&hadc);
    HAL_ADC_PollForConversion(&hadc, 100);
    counts = HAL_ADC_GetValue(&hadc);
    HAL_ADC_Stop(&hadc);
    return(counts * ADC_RES); // ADC_RES = 3.3 / 4096
}
```

Dado que el sistema genera una interrupción cada 10 us y la lectura del ADC se realiza cada 10 interrupciones, la frecuencia efectiva de muestreo es:

$$f_s = \frac{1}{10ms} = 100Hz$$

Esta frecuencia resulta insuficiente para la aplicación de este trabajo practico ya que se estaria muestreando audio con una frecuencia máxima de 18kHz dada por el microfono. De esta forma, para cumplir con el teorema de Nyquist, la frecuencia de muestreo deberia ser de al menos 36 kHz para poder capturar el contenido sin aliasing. Para adaptar el sistema a procesamiento de audio, se propone la siguiente modificación en my_lib.h:

```
#define Timer_Period_us 10  //
#define _BasePeriod(x_us) (((x_us) * (CORE_CLK/1000000)) /
BASE_PRESCALER - 1)
#define ADC_Sampling_Period ((uint32_t) 2)//fs termina siendo
50khz
```

Luego, establezco a ADC_Sampling_Period a 2 de modo que se realice una lectura de ADC en cada interrupción. En la función TIM3_Config(), se reemplaza el uso de Timer_Period por Timer_Period_us y con esta configuración, el sistema genera interrupciones cada 10 µs, y por lo tanto permite una frecuencia de muestreo de 50 kHz. En my_lib.h se debe redefinir el macro _BasePeriod para que trabaje en microsegundos y ajustar el nuevo período. Luego, el valor de sampling se sigue utilizando como antes, pero ahora se reinicia cada 10 µs, por lo que:

$$f_s = \frac{1}{10\mu s \cdot ADC\_Sampling\_Period} = 50kHz$$

De esta forma obtenemos una frecuencia de muestreo de 50kHz sin cambiar el ciclo principal en main.c.

Calculo de Filtro

Para implementar un filtro FIR (Finite Impulse Response) en el microcontrolador STM32F401RE, es necesario en primer lugar calcular los coeficientes o "taps" del filtro. En este caso, los coeficientes fueron generados utilizando un script en Python proporcionado por el profesor.

La frecuencia de muestreo establecida en el sistema es de 50 kHz, la cual debe ser respetada durante el diseño del filtro para evitar inconsistencias temporales. Se fijó una frecuencia de corte de 12 kHz, de forma que se logre un efecto audible cuando la señal pase a la etapa posterior.

En la búsqueda de recomendaciones prácticas para la implementación de filtros FIR en microcontroladores STM32, se consultó bibliografía oficial de STMicroelectronics. En particular, se analizó el documento DM00605584, donde se describe la implementación de filtros FIR utilizando la unidad FMAC (Filter Math Accelerator) en micros de la familia STM32G4. Aunque el STM32F401RE no dispone de dicho periférico, el documento presenta lineamientos generales válidos para cualquier MCU de ST, incluyendo aspectos sobre rendimiento, tamaño de bloque y número máximo de coeficientes recomendado.

En dicho documento se sugiere que, en aplicaciones embebidas sin acelerador dedicado, el número de coeficientes del filtro se mantenga dentro de un rango razonable, típicamente entre 11 y 51 taps, según la capacidad de procesamiento y los requerimientos de atenuación en las bandas de transición. Teniendo en cuenta que el filtrado se realiza en tiempo real, una mayor cantidad de coeficientes implica una convolución más extensa y por ende una mayor carga para el CPU, lo cual puede afectar otras tareas concurrentes del sistema. Por este motivo, se optó por un filtro FIR de 11 coeficientes (orden 10), lo cual ofrece un equilibrio entre selectividad y eficiencia computacional.

Los coeficientes utilizados para este filtro son los siguientes:

[1.87032941e-18, 1.26981175e-02, -2.48019135e-02, -6.37871498e-02, 2.76018100e-01, 5.99745691e-01, 2.76018100e-01, -6.37871498e-02, -2.48019135e-02, 1.26981175e-02, 1.87032941e-18]

Y analisis de polos y respuesta en frecuencia proporcionada por el script se puede ver en la figura 5.

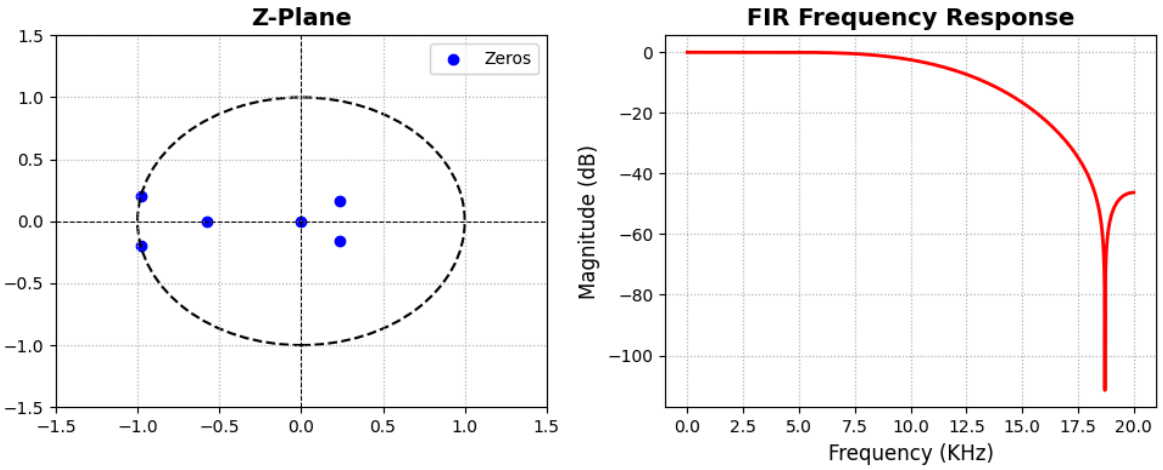


Figura 5: Analisis de polos y respuesta en frecuencia del filtro FIR

Implementación

Para implementar el filtro digital diseñado en el apartado anterior, se reutilizará parte del codigo del proyecto FIR modificando los coeficientes por los calculados con el script.

```
#define FILTER_TAP_NUM 11
const float firCoeffs[FILTER_TAP_NUM] = {
          1.87032941e-18, 1.26981175e-02, -2.48019135e-02,
          -6.37871498e-02, 2.76018100e-01, 5.99745691e-01,
          2.76018100e-01, -6.37871498e-02, -2.48019135e-02,
          1.26981175e-02, 1.87032941e-18
};
```

En el main la utilización quedaria de la siguiente manera.

```
int main(void){
float new_sample, filtered_value;
    // Hardware Initialize
    Hw_Init();
    // Sampling Tick
    sampling = ADC_Sampling_Period;
    // Infinite Loop
    while (1)
    {
        if (sampling == 0)
        {
           // Assume we read a new ADC value here
            new_sample = ADC_Read();
            // Apply FIR filter
            filtered_value = FIR_Filter(new_sample);
```

```
        }
    }
}
```

Donde en la función FIR_Filter esencialmente se hace la convolucion del dato con los taps del filtro.

PWM

Preliminares

Para realizar la modulación en ancho de pulso (PWM) fue necesario adaptar la salida del filtro FIR como entrada del controlador del ciclo de trabajo. Esta sección detalla el análisis de los coeficientes del filtro, la estimación del rango de salida y la implementación de la conversión al duty cycle correspondiente.

El filtro utilizado posee 11 coeficientes diseñados en punto flotante. Se calculó la suma de los valores absolutos de estos coeficientes con el objetivo de estimar el valor máximo que puede alcanzar la salida del filtro FIR ante una entrada de amplitud constante. El resultado de esta suma fue aproximadamente:

$$\sum_{i=0}^{10} |h[i]| \simeq 0.999999984 \approx 1$$

Esto indica que el filtro está efectivamente normalizado. Por lo tanto, la salida del filtro puede alcanzar, como máximo, el mismo valor que la entrada, en caso de que todas las muestras estén alineadas constructivamente.

Para poder convertir la señal filtrada en un valor de dutty cycle del PWM, se utilizó la siguiente ecuación.

$$\delta = \frac{filtered\_value}{3.3} \cdot 100$$

Implementación

En el programa principal se implementara la ecuación de ajuste del dutty y la posterior modulación en PWM.

```
while (1){
     if (sampling == 0){
        // Assume we read a new ADC value here
        new_sample = ADC_Read();
        // Apply FIR filter
        filtered_value = FIR_Filter(new_sample);
        dutty = (filtered_value/3.3);
        TIM2->CCR1=_PWMDutty(dutty,_PWMPeriod(Carrier_Period));
     }
```

Además de esto, se ajustó el periodo de la portadora de la siguiente manera

```
#define Carrier_Period ((uint32_t) 10) // 1 [us]
```

De esta manera, se toma un periodo de la portadora de 10us, resultando en una frecuencia de 100kHz. Siguiendo los lineamientos de Microchip (Technology Inc, 2020), esta frecuencia debería ser suficiente para poder filtrar la onda cuadrada sin perder el espectro de la señal filtrada por el filtro FIR.

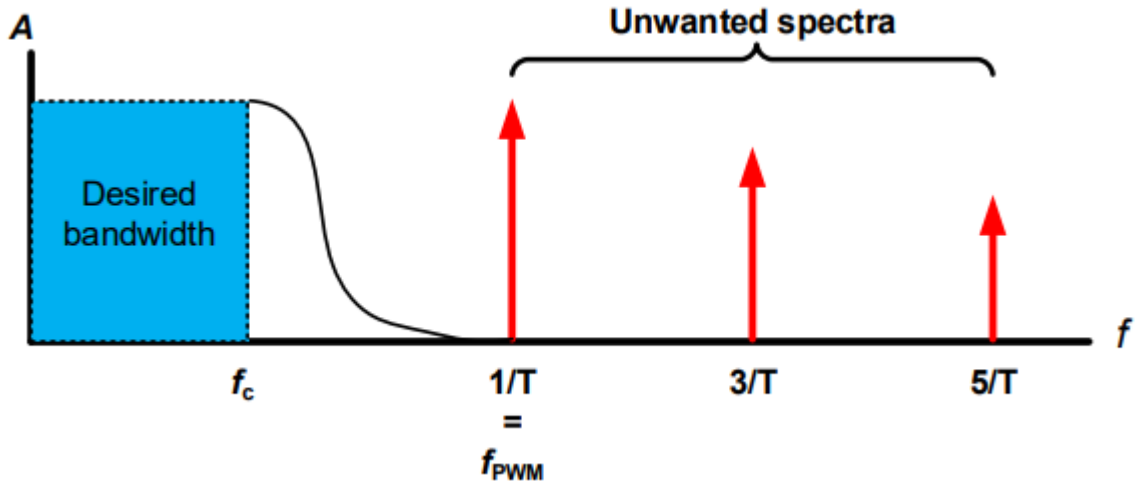
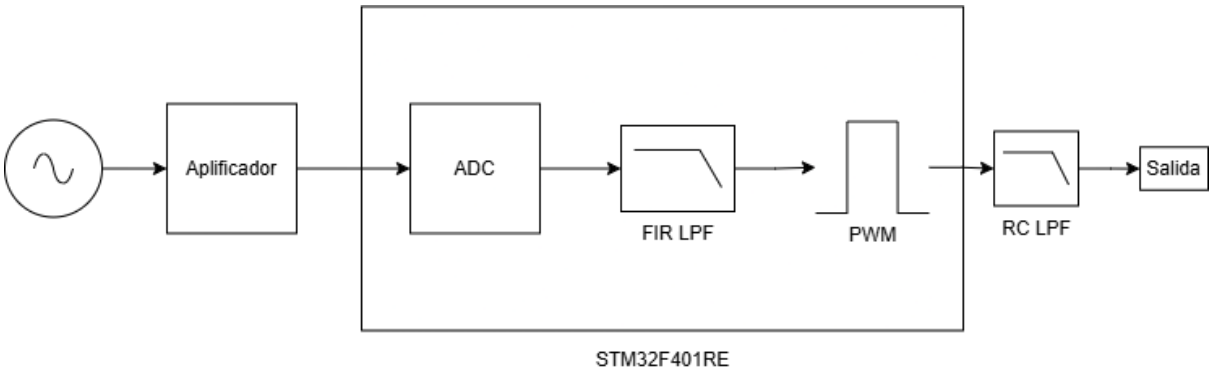


Diagrama de Bloques

A continuación se expone el diagrama de bloques del sistema.



Posibles Mejoras

Cambio de Fc del Filtro con un Potenciometro

Una mejora relevante para el sistema implementado consiste en permitir el ajuste dinámico de la frecuencia central del filtro FIR pasabanda, mediante una interfaz analógica sencilla y de bajo costo. Esta funcionalidad no solo amplía el rango de aplicación del dispositivo, sino que también introduce una interacción directa con el usuario, útil en contextos donde se requiera adaptar el filtrado en función del entorno acústico o de criterios subjetivos de audición.

Desde el punto de vista teórico, el desplazamiento de la frecuencia central de un filtro FIR se logra a través de la modulación espectral de sus coeficientes. A partir de un conjunto de coeficientes base h[n], correspondiente a un filtro pasa bajo o pasa banda centrado en una frecuencia fija, se puede obtener un nuevo filtro hmod[n] desplazado espectralmente mediante la multiplicación por una señal cosenoidal:

$$h_{mod}[n] = h[n] \cdot cos(2\pi f_c n T_s)$$

Donde fc representa la frecuencia de desplazamiento deseada y Ts el período de muestreo. Este procedimiento es equivalente a trasladar el espectro del filtro en el dominio de la frecuencia, sin modificar su forma original. En términos computacionales, esta operación puede realizarse en tiempo real con un bajo costo de procesamiento, especialmente si se cuenta con una unidad de punto flotante, como la disponible en el microcontrolador STM32F401RE.

Para la implementación práctica de esta mejora, se propone el uso de un potenciómetro conectado a una entrada analógica del microcontrolador. El potenciómetro actúa como un divisor resistivo, cuya salida se mide a través del ADC integrado. La tensión leída se

convierte en un valor digital proporcional a una frecuencia de desplazamiento fc, la cual se utiliza para modular los coeficientes del filtro en tiempo real. Esta arquitectura permite que el usuario controle manualmente la frecuencia central del filtro girando el potenciómetro, lo que se traduce en un corrimiento del espectro del filtro aplicado a la señal de entrada.

Optimización del DSP

En el sistema actual, la lectura de datos provenientes del ADC se realiza mediante una rutina de polling o por interrupciones periódicas, tras verificar manualmente el estado del periférico desde el programa principal. Si bien esta estrategia es funcional para frecuencias de muestreo bajas (por ejemplo, 100 Hz), su escalabilidad es limitada en contextos donde se requiere capturar señales de mayor velocidad, como lo es el caso del trabajo práctico, donde la frecuencia de muestreo es de 50kHz.

Cuando se incrementa la frecuencia de muestreo, la intervención directa del CPU en cada conversión del ADC implica un consumo considerable de ciclos de instrucción. Esto ocurre porque cada operación de muestreo requiere al menos iniciar la conversión, esperar que finalice y leer el resultado. Sumado a esto, tambien se está empleando tiempo de procesamiento al implementar un filtro digital en el CPU (ya que este microcontrolador no posee FMAC).

A frecuencias como 50kHz (una conversión cada 20 µs), este enfoque puede resultar inviable si la ejecución combinada de lectura y procesamiento excede ese intervalo, produciendo jitter en el muestreo, pérdida de datos o inestabilidad en la respuesta temporal del sistema (Microelectronics, 2019).

Para asegurar estabilidad temporal en muestreos de alta velocidad, es recomendable delegar las tareas de adquisición a mecanismos automáticos como el DMA (Direct Memory Access). Este periférico permite transferir datos directamente desde el ADC a la memoria dinamica sin intervención del CPU, lo que garantiza que cada conversión quede almacenada a tiempo, incluso en sistemas con múltiples interrupciones o procesamiento concurrente.

Adicionalmente, existen microcontroladores que integran aceleradores hardware específicamente diseñados para procesamiento digital de señales. Un ejemplo destacado es el FMAC (Filter Math Accelerator) incluido en la familia STM32G4, que permite aplicar filtros FIR e IIR directamente en hardware sin intervención del núcleo principal. Este tipo de periférico permite implementar soluciones de procesamiento digital en tiempo real con mucha mayor eficiencia, al liberar completamente al CPU de operaciones matemáticas repetitivas.

En contraste, el STM32F401RE no cuenta con aceleradores dedicados como FMAC, pero sí ofrece mecanismos como trigger externo del ADC y el DMA, que constituyen una solución intermedia altamente eficiente. En este esquema, un temporizador (por ejemplo, TIM3) se configura para generar un evento a una frecuencia deseada (como 50 kHz), el cual dispara automáticamente el inicio de conversión del ADC. La conversión completada es luego

transferida mediante DMA a un buffer en memoria. El CPU solo interviene al finalizar el llenado del buffer, pudiendo procesar los datos por bloques (batch processing), lo cual optimiza el rendimiento general del sistema y garantiza un muestreo regular, sin jitter.