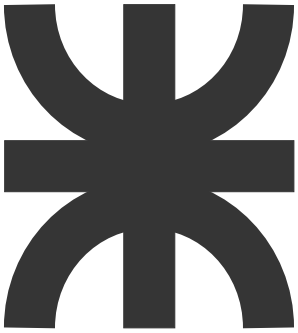
U.T.N. - Facultad Regional Paraná



Técnicas Digitales III - TPN°3
Implementación de Filtros FIR e IIR

Alumnos:

- Perissutti, Gianfranco Adriano
- Ubiedo, Lautaro Ezequiel

Profesores:

- Ing. Alejandro Dachary
- Ing. Gustavo A. Yarce

Fecha de Entrega: 04/08/2023

Paraná - Entre Ríos

Técnicas Digitales III - TPN° 3

Índice

1 Diseño de Filtros

Para este trabajo práctico se requieren implementar 6 filtros, un pasabajos, un pasaaltos y un pasabanda, los tres se diseñarán tanto con respuesta al impulso finita como con respuesta al impulso infinita. Para diseñar estos filtros se empleó la app de MATLAB *Filter Designer* ya que nos permite exportar los coeficientes de la ecuación en diferencia para poder emplearlos posteriormente en el programa del microcontrolador.

La facilidad que nos otorga *Filter Designer* es que podemos exportar los coeficientes de punto flotante de precisión simple como archivos *header (.h)* lo que nos permite usarlos como librerías de C las cuales incorporaremos luego en el programa del microcontrolador. Esto nos da mayor modularidad a la hora de programar ya que solo se debe cambiar el archivo .h del filtro para probar otros coeficientes, ya que el programa principal del microcontrolador no necesita cambio alguno.

Para los filtros decidimos utilizar como frecuencias de corte 1[KHz] y 3.5[KHz]. En los filtros pasabajos las frecuencias de corte son de 1[KHz], en los filtros pasaaltos son de 3.5[KHz] y en los filtros pasabandas las bandas de paso van desde 1[KHz] hasta 3.5[KHz]. Debido a esta selección de frecuencias de corte decidimos emplear una frecuencia de muestreo de 20[KHz], ya que contemplamos emplear señales de hasta 5[KHz] para hacer las pruebas por lo que al menos deberíamos tener una frecuencia de muestreo 4 veces superior a la mayor frecuencia para así evitar los problemas de aliasing.

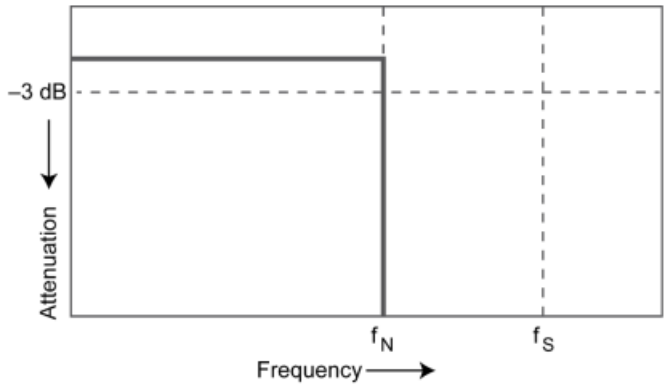


Figura 1: Frecuencia de Muestreo con Respuesta Plana.

Si bien la frecuencia de Nyquist dice que la frecuencia de muestreo debe ser al menos el doble de la mayor frecuencia, esto solo aplica cuando la respuesta del filtro es totalmente plana y como esto solo sucede en la teoría, en la práctica se emplea una frecuencia de muestreo de al menos 4 veces superior a la mayor frecuencia, esto es un concepto de diseño muy usado en la industria.

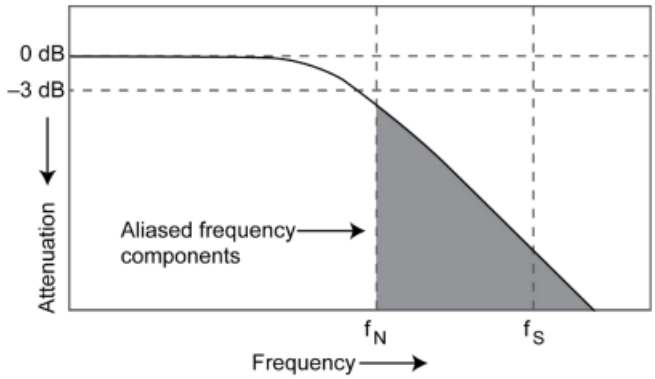


Figura 2: Frecuencia de Muestreo en la Práctica.

1.1. Filtros FIR

Para los filtros del tipo FIR se empleo el diseño mediante la ventana de Hamming. Mediante el uso de la app *Filter Designer* podemos ver la respuesta en frecuencia de los filtros.

El filtro pasabajos, teniendo una frecuencia de corte de -6[dB] en 1[KHz], resulta ser de orden 30 y su respuesta en frecuencia es la siguiente.

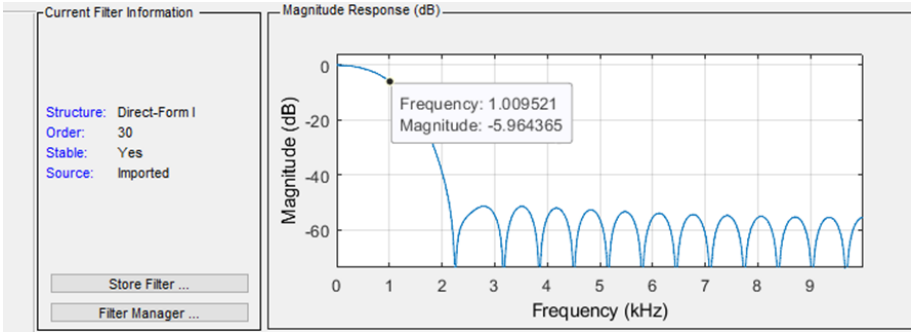


Figura 3: Magnitud de la Respuesta en Frecuencia del LPF FIR.

El filtro pasabanda, teniendo una frecuencia de corte inferior de -6[dB] en 1[KHz] y una frecuencia de corte superior de -6[dB] en 3.5[KHz], resulta ser de orden 24 y su respuesta en frecuencia es la siguiente.

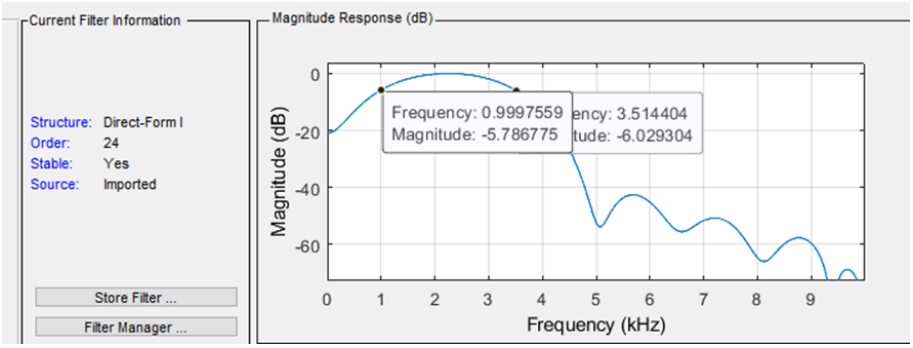


Figura 4: Magnitud de la Respuesta en Frecuencia del BPF FIR.

El filtro pasaalto, teniendo una frecuencia de corte de -6[dB] en 3.5[KHz], resulta ser de orden 20 y su respuesta en frecuencia es la siguiente.

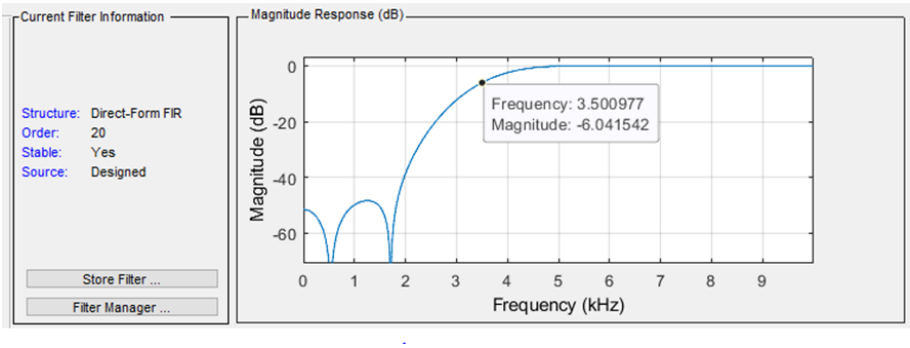


Figura 5: Magnitud de la Respuesta en Frecuencia del HPF FIR.

1.2. Filtros IIR

Los filtros tipo IIR deben ser al menos de orden 2, pero debido a nuestra elección de las frecuencias de corte los tres filtros serán de orden superior. Para el diseño de los filtros se decidió por emplear la topología de Butterworth.

El filtro pasabajos, teniendo una frecuencia de corte de -6[dB] en 1[KHz], resulta ser de orden 3 y su respuesta en frecuencia es la siguiente.

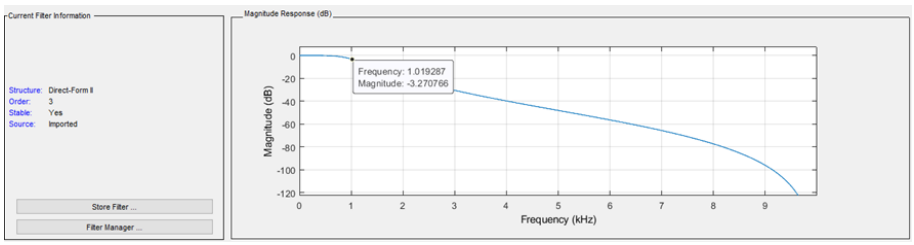


Figura 6: Magnitud de la Respuesta en Frecuencia del LPF IIR.

El filtro pasabanda, teniendo una frecuencia de corte inferior de -6[dB] en 1[KHz] y una frecuencia de corte superior de -6[dB] en 3.5[KHz], resulta ser de orden 6 y su respuesta en frecuencia es la siguiente.

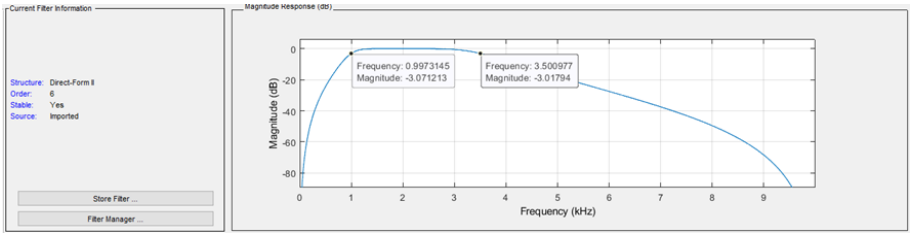


Figura 7: Magnitud de la Respuesta en Frecuencia del BPF IIR.

El filtro pasaalto, teniendo una frecuencia de corte de -6[dB] en 3.5[KHz], resulta ser de orden 5 y su respuesta en frecuencia es la siguiente.

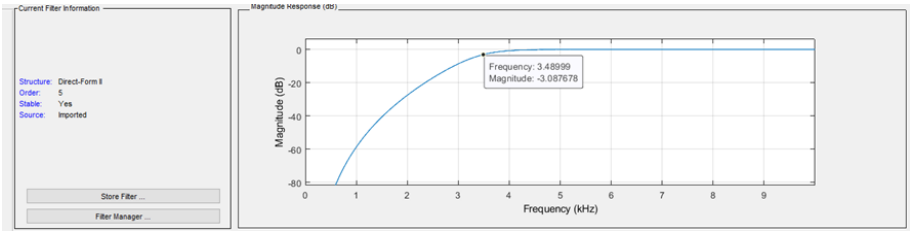


Figura 8: Magnitud de la Respuesta en Frecuencia del HPF IIR.

Los coeficientes de los filtros se exportaron como headers de c para integrarlos al codigo facilmente, sus valores fueron cuantizados a flotante de precision simple ya que el micro cuenta con una FPU integrada.

2 Programación del Microcontrolador

Debido a que vamos a emplear una frecuencia de muestreo relativamente alta decidimos emplear la placa de desarrollo FF4VE Blackboard con el microcontrolador STM32F407VET6 que utiliza un core Arm-Cortex M4.

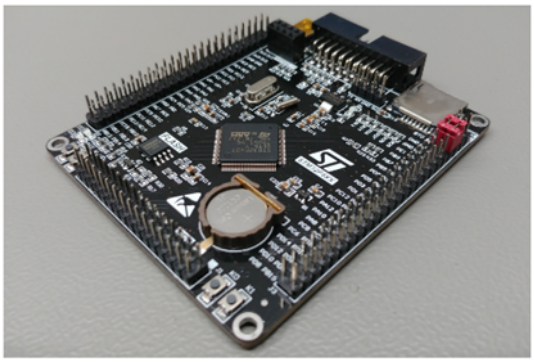


Figura 9: Microcontrolador STM32F407VET6.

2.1. Configuracion del ADC

Decidimos emplear esta placa ya que su ADC es de 12 bits y permite una frecuencia de muestreo de hasta 2.4[MSpS]. Además, posee un DAC de 12 bits interno que nos facilita las pruebas ya que no se necesita hacer un circuito externo.

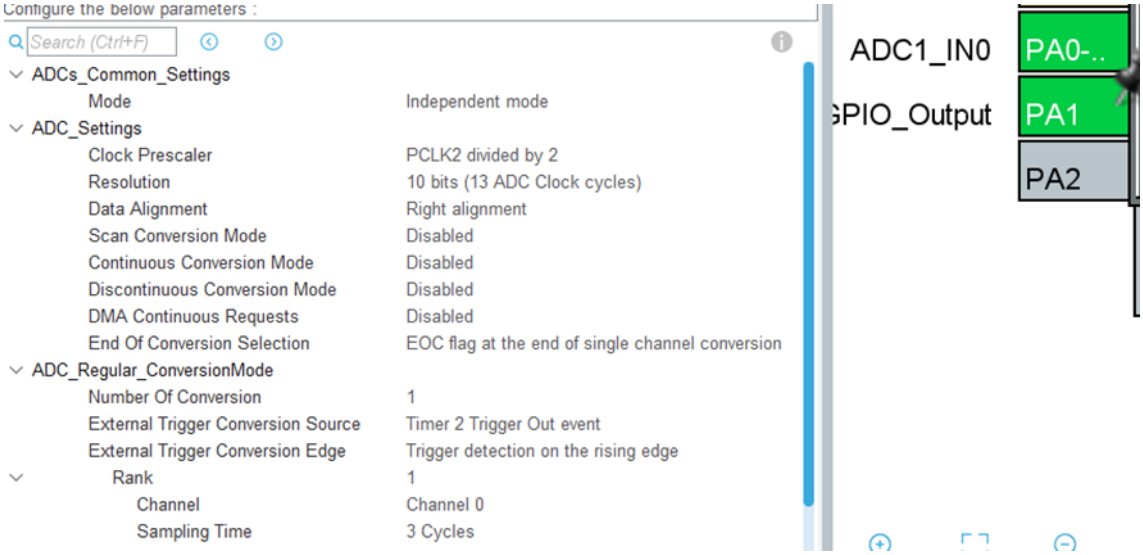


Figura 10: Configuracion ADC.

2.2. Configuracion del timer

En la configuración del timer en STM32Cube, comenzamos estableciendo el evento de actualización, también conocido como evento de overflow en otros microcontroladores. Este evento de overflow es esencial para nuestro funcionamiento, ya que determina el ciclo de trabajo y la frecuencia de nuestro sistema. Con una frecuencia del bus del timer de 84 MHz y la frecuencia deseada del evento de 20 kHz, que define la frecuencia de muestreo para nuestros filtros, realizamos un cálculo preciso. Utilizamos el valor máximo al que cuenta el timer en conjunción con la frecuencia del bus para lograr la frecuencia deseada.

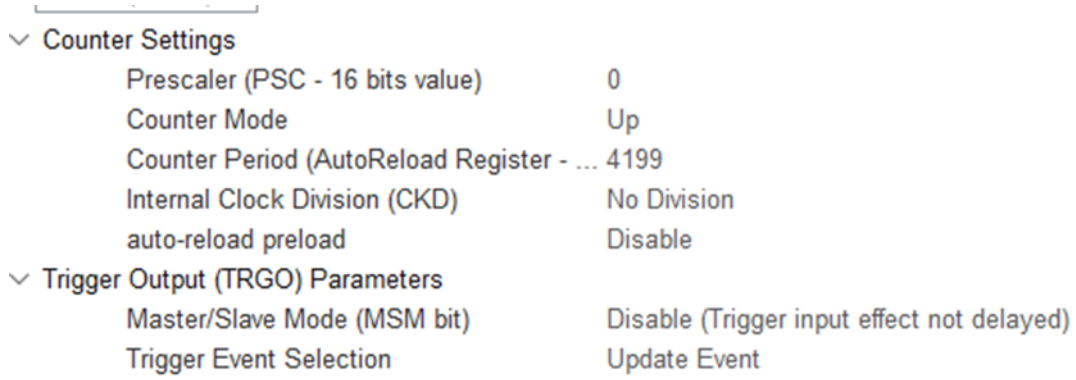


Figura 11: Configuracion timer.

2.3. Configuracion de la comunicacion serie

En el contexto de la comunicación serie en STM32Cube, hemos optado por utilizar la interfaz USB FS (Full Speed). Esta configuración nos permite establecer una conexión USB virtual COM utilizando el middleware proporcionado por STM. Al habilitar esta funcionalidad, logramos una comunicación bidireccional confiable y eficiente entre nuestro microcontrolador y el host.

Figura 12: Configuracion usb.

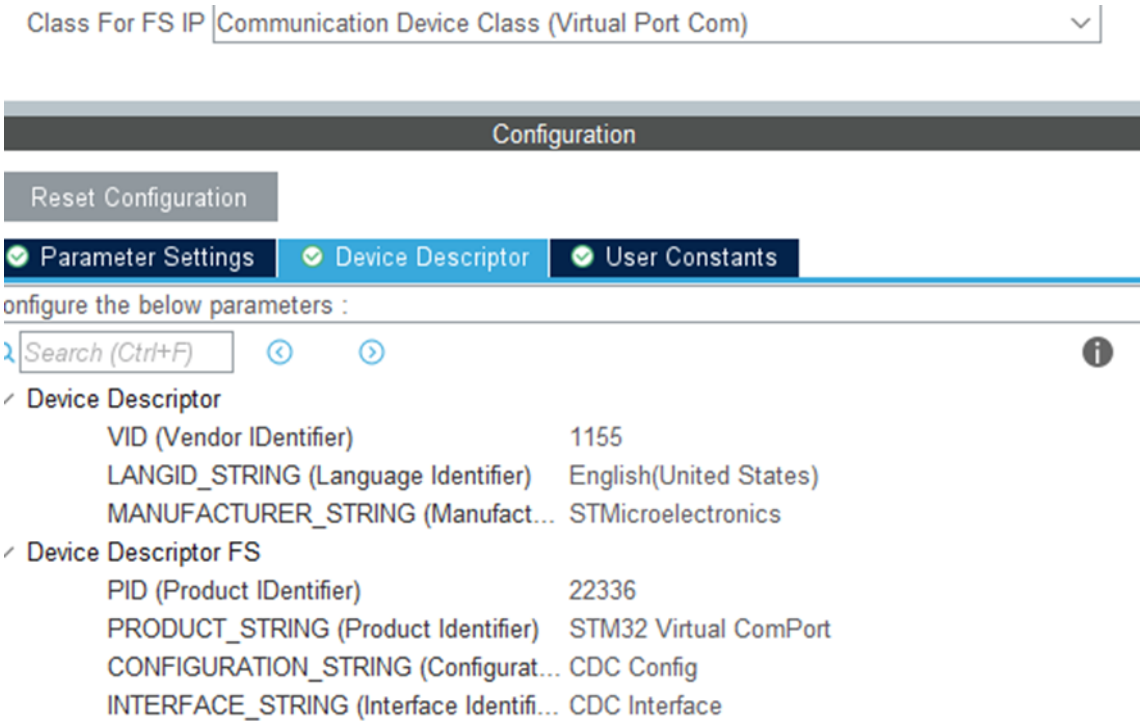


Figura 13: Configuracion usb.

2.4. DAC

Configurado a 8 bits.

3 Codigo:

Descripción del código del trabajo práctico En la etapa inicial, se procede a la elaboración y establecimiento de una librería específica destinada al cálculo algebraico del filtro.

```c
#include "filtros_digitales.h"

void IIR_filter(float *x, float *y,float *a, float *b, unsigned int order)
{
    y[0]= 0;
      for(unsigned int i = 0; i<(order+1); i++)
            {
                y[0] += x[i]*b[i] ;
            }
            for(unsigned int i = 1; i<(order+1);i++)
            {
                y[0] -= y[i]*a[i];
            }

    for(unsigned int i = (order+1); i>0; i--)
        {
            y[i] = y[i-1];
            x[i] = x[i-1];
        }



}
void FIR_filter(float *x, float *y,float *b, unsigned int order)
{
    y[0]= 0;
     for(unsigned int i = 0; i<(order+1); i++)
            {
                y[0] += x[i]*b[i] ;
            }

      for(unsigned int i = (order+1); i>0; i--)
        {
            x[i] = x[i-1];
        }
```

Figura 14: Libreria de filtros.

Acto seguido, en la sección principal (Main) del programa, se desarrolla un proceso que se subdivide en tres componentes fundamentales.

La primera de estas partes abarca la configuración y ajuste de la totalidad de los periféricos involucrados en el sistema, así como la activación de indicadores (flags) pertinentes. Además, se procede a la declaración y asignación de variables a nivel local y global, considerando las necesidades específicas del proceso en cuestión.

```
if(LOW_PASS == filter_band)
{
    for(unsigned int i= 0; i<MAX_SIZE;i++)
      {
         y[i]=0.0;
         x[i]=0.0;
      }
}
while((LOW_PASS == filter_band) && (FIR == filter_type) && (!send_data))
{
    if(adc_conversion_complete)
    {
        x[0] = ((adc_val-512)*1.15)/1024.0f;
        FIR_filter(x, y,b_low_FIR,order_low_FIR);
        output_volt = ((y[0]*255-127)/1.15);
        HAL_DAC_SetValue(&hdac,DAC1_CHANNEL_1, DAC_ALIGN_8B_R,output_volt );
        adc_conversion_complete= 0;
    }
}
```

Figura 15: Parte del loop principal.

Segunda parte importante: Núcleo central iterativo, en el cual se implementan estructuras condicionales que se valen de las variables booleanas previamente mencionadas para efectuar consultas determinantes acerca del estado operativo actual requerido en el microcontrolador.

```
if(LOW_PASS == filter_band)
{
    for(unsigned int i= 0; i<MAX_SIZE;i++)
      {
         y[i]=0.0;
         x[i]=0.0;
      }
}
while((LOW_PASS == filter_band) && (FIR == filter_type) && (!send_data))
{
    if(adc_conversion_complete)
    {
        x[0] = ((adc_val-512)*1.15)/1024.0f;
        FIR_filter(x, y,b_low_FIR,order_low_FIR);
        output_volt = ((y[0]*255-127)/1.15);
        HAL_DAC_SetValue(&hdac,DAC1_CHANNEL_1, DAC_ALIGN_8B_R,output_volt );
        adc_conversion_complete= 0;
    }
}
```

Figura 16: Parte del loop principal.

```
if(send_data)
{
    for(unsigned int i = 0; i<BUFFER_SIZE; i++)
    {
       sprintf(msg_data, "%2.3f \n", buffer_data[i]);
       CDC_Transmit_FS(msg_data,strlen(msg_data));
       HAL_Delay(50);
    }
    send_data = 0;
}
```

Figura 17: Parte del loop principal.

En la siguiente figura se observa el cambio de estado de las banderas dentro de la funcion de lectura completada del ADC.

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{

    HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_1);
    adc_val = HAL_ADC_GetValue(&hadc1);
    adc_conversion_complete= 1;
    if(get_data==1)
    {
        buffer_data[buffer_index]= y[0];
        buffer_index++;
        if(buffer_index >= BUFFER_SIZE)
        {
            get_data= 0;
            send_data = 1;
            buffer_index= 0;
        }
    }

}
```

Figura 18: Cambio de estados de booleanos.

Tercera etapa del segmento principal: Comunicación serial. En esta sección, se establece una conexión con el puerto USB mediante la implementación de funciones intrínsecas a las interrupciones. A través de esta interacción, se efectúa la manipulación de los indicadores (flags), adaptándolos conforme a las solicitudes específicas formuladas por el usuario.

```
void CDC_ReceiveCallBack(uint8_t *buf, uint32_t len)
{
    char *buffer_recibido = buf;
    char* msg;
    if((strcmp(buffer_recibido,"data_req") == 0) && (!send_data))
    {
        get_data = 1;
        msg = "Obteniendo datos \n\r\0";
        CDC_Transmit_FS(msg,strlen((char* )msg));
    }
    if((strcmp(buffer_recibido,"IIR_req") == 0) && (!send_data))
    {
        filter_type = IIR;
        msg ="Se cambio a IIR \n\0";
        CDC_Transmit_FS(msg,strlen((char* )msg));
    }
    if((strcmp(buffer_recibido,"FIR_req") == 0) && (!send_data))
    {
        filter_type = FIR;
        msg ="Se cambio a FIR \n\0";
        CDC_Transmit_FS(msg,strlen((char* )msg));
    }
    if((strcmp(buffer_recibido,"LPf_req") == 0) && (!send_data))
    {
        filter_band = LOW_PASS;
        msg ="Se cambio a pasabajos";
        CDC_Transmit_FS(msg,strlen((char* )msg));
    }
```

Figura 19: Comunicacion y recepcion serie.

4 Adquisición de Datos

La operación de interacción del usuario con el sistema se ejecuta mediante un programa implementado en el lenguaje de programación Python. El proceso se inicia al consultar, en primer término, el puerto de destino al cual establecer la conexión. Una vez que la conexión se ha establecido de manera exitosa, se despliega un menú de opciones que permiten al usuario interactuar con los diversos filtros. Dicho menú se ajusta de acuerdo a los requisitos específicos establecidos en el marco del trabajo práctico.

```python
puerto_com = input("Ingrese el número del puerto COM (por ejemplo, 6 para COM6): ")
puerto_str = f'COM{puerto_com}'
# Crea un objeto Serial con la configuración adecuada para el puerto COM elegido
try:
    puerto = serial.Serial(puerto_str, baudrate=57600, timeout=1)
except serial.SerialException as e:
    print(f"Error al abrir el puerto {puerto_str}: {e}")
    fin_exec = 1
finally:

    while(fin_exec <= 0):
        print("Menu de comandos:")
        print("-Ingrese 'F' para cambiar a filtro FIR" )
        print("-Ingrese 'I' para cambiar a filtro IIR" )
        print("-Ingrese 'LP' para cambiar a filtro pasabajos" )
        print("-Ingrese 'BP' para cambiar a filtro pasabanda" )
        print("-Ingrese 'HP' para cambiar a filtro pasaaltos" )
        print("-Ingrese 'M' para guardar 250 valores en archivo de texto")
        print("-Ingrese 'S' para salir")

        comando = input("Ingrese un comando:")

        if(comando == 'M'):
```

Figura 20: Codigo de la interfaz del usuario

```
Ingrese el número del puerto COM (por ejemplo, 6 para COM6): 6
Menu de comandos:
-Ingrese 'F' para cambiar a filtro FIR
-Ingrese 'I' para cambiar a filtro IIR
-Ingrese 'LP' para cambiar a filtro pasabajos
-Ingrese 'BP' para cambiar a filtro pasabanda
-Ingrese 'HP' para cambiar a filtro pasaaltos
-Ingrese 'M' para guardar 250 valores en archivo de texto
-Ingrese 'S' para salir
Ingrese un comando:F
Se cambio a FIR
```

Figura 21: Interfaz del usuario

5 Posprocesamiento de Datos

Los datos se guardan en un archivo de texto separados por un delimitador de salto de linea. Se pueden leer en multiples programas matematicos como asi tambien de procesamiento de datos numericos y estadisticos. Decidimos no guardarlo en forma binaria para poder leer el archivo al abrirlo.

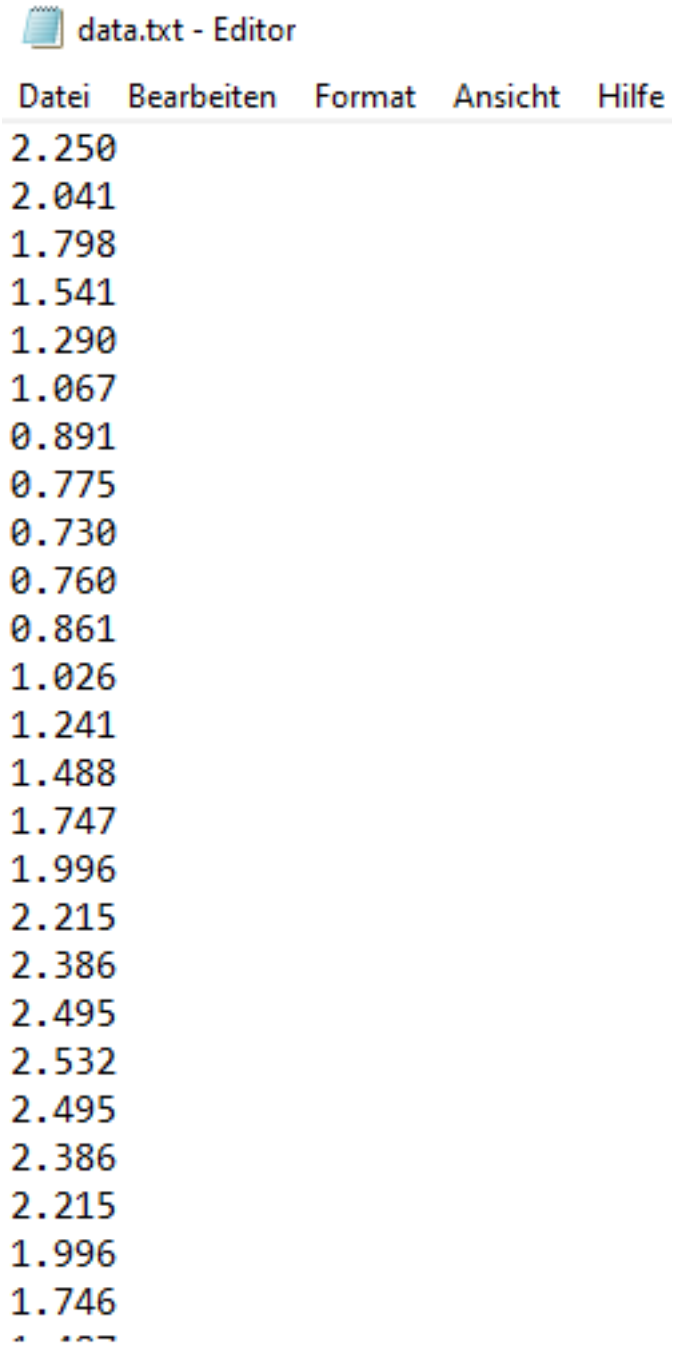


Figura 22: Archivo escrito con los datos de salida

6 Pruebas

Nos encontramos con un problema principal durante el desarrollo en relación a la atenuación total del valor de corriente continua en las señales de salida. Esto fue especialmente notorio en los filtros pasabanda y pasaaltos.

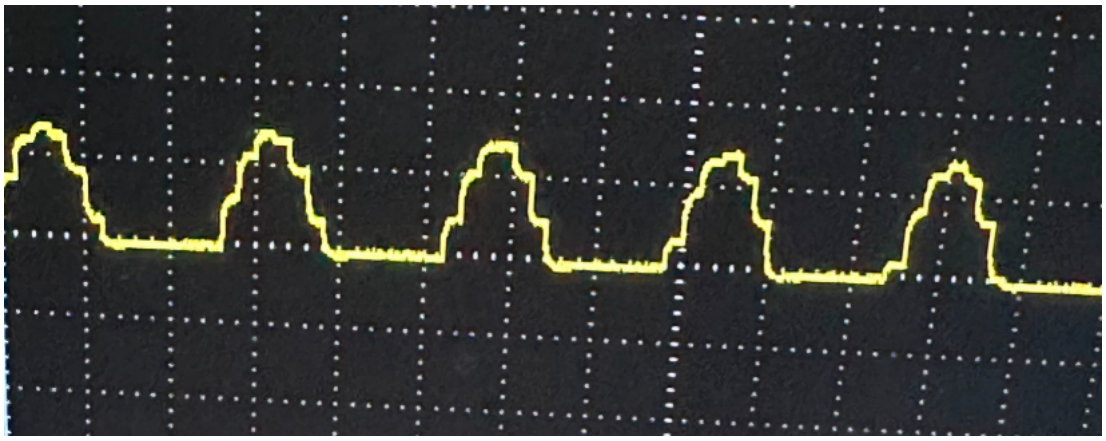


Figura 23: Señal de salida sin continua

Resolvimos este problema ajustando el valor medido por el ADC. Primero, eliminamos el valor de corriente continua promedio antes de aplicar el filtro. Después de calcular el filtro, reintegramos el valor de continua a la señal filtrada. Esto solucionó la atenuación excesiva en los filtros pasabanda y pasaaltos.

7 Conclusion final

En resumen, el uso de MATLAB para implementar filtros digitales resultó efectivo. Sin embargo, enfrentamos desafíos, como la atenuación total del componente de corriente continua en los filtros pasabanda y pasaaltos. Solucionamos esto ajustando los valores medidos por el ADC, eliminando y luego reintegrando la corriente continua. Esto resalta la importancia de un análisis profundo y soluciones creativas en el procesamiento de señales y filtros digitales. Los filtros funcionaron como se esperaba con una tolerancia del 2 % de las frecuencias de corte calculadas.