Proyecto final medidor de ruido de linea

Objetivo:

Realizar una medición del ruido de línea de alimentación (220v 50Hz), identificando la potencia de ruido y en qué frecuencias se produce.

Tipo de ruidos:

El ruido eléctrico es una señal de interferencia eléctrica no deseada, se lo puede clasificar según su origen como generados por la naturaleza, o generados por el hombre. Por lo general las generados por la naturaleza son aleatorios y de poco duración, como por ejemplo un rayo. Los generados por el hombre son en general debido a equipos electrónicos o eléctricos, como por ejemplo un motor, o el ruido de un fuente de alimentación. Estos últimos pueden se aperiódicos o tener un cierto grado de periocidad, en este proyecto nos `vamos a dedicar a estos últimos.

Efectos de un ruido eléctrico:

Las interferencias electromagnéticas se producen normalmente en entornos industriales y pueden influir negativamente en las señales de comunicación y/o alimentación. Algunos ejemplos son:

1 - Problemas de vídeo en pantallas, como rayas y barras.

2 - Problemas de audio como zumbidos.

3 - Interferencia de la PC y mal funcionamiento, incluyendo paradas y problemas de red aleatorios.

4 - Caídas de datos y brechas de descarga.

5 - Problemas de calidad de energía con equipos electrónicos como caídas de tensión, fallas de equipo, reinicios, etc.

6 - Lecturas y datos imprecisos del ruido de la señal.

Módulo ACS712:

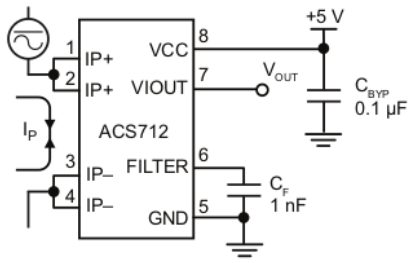
Para medir la corriente se usa el circuito integrado ACS 712 de Allegro Inc. El cual mediante efecto Hall censa la corriente desde CC hasta unos 80KHz.
Las características principales del circuito integrado son:

FEATURES AND BENEFITS

- Low-noise analog signal path
- Device bandwidth is set via the new FILTER pin
- 5 µs output rise time in response to step input current
- 80 kHz bandwidth
- Total output error 1.5% at T_A = 25°C
- Small footprint, low-profile SOIC8 package
- 1.2 mΩ internal conductor resistance
- 2.1 kVRMS minimum isolation voltage from pins 1-4 to pins 5-8
- 5.0 V, single supply operation
- 66 to 185 mV/A output sensitivity
- Output voltage proportional to AC or DC currents
- Factory-trimmed for accuracy
- Extremely stable output offset voltage
- Nearly zero magnetic hysteresis
- Ratiometric output from supply voltage

La aplicación típica, y la configuración de pines se muestra a continuación:

Typical Application



Application 1. The ACS712 outputs an analog signal, V_{OUT}, that varies linearly with the uni- or bi-directional AC or DC primary sampled current, I_P, within the range specified. C_F is recommended for noise management, with values that depend on the application.

Este tipo de dispositivo viene en tres versiones en que se diferencian en el limite máximo de corriente medida y la sensibilidad de la medición:

SELECTION GUIDE

Part Number	Packing*	T_A (°C)	Optimized Range, I_P (A)	Sensitivity, Sens (Typ) (mV/A)
ACS712ELCTR-05B-T	Tape and reel, 3000 pieces/reel	–40 to 85	±5	185
ACS712ELCTR-20A-T	Tape and reel, 3000 pieces/reel	–40 to 85	±20	100
ACS712ELCTR-30A-T	Tape and reel, 3000 pieces/reel	–40 to 85	±30	66

*Contact Allegro for additional packing options.

Se utilizará la versión que mide hasta 30A.

Los valores máximos de funcionamiento son:

ABSOLUTE MAXIMUM RATINGS

Characteristic	Symbol	Notes	Rating	Units
Supply Voltage	V_{CC}		8	V
Reverse Supply Voltage	V_{RCC}		–0.1	V
Output Voltage	V_{IOUT}		8	V
Reverse Output Voltage	V_{RIOUT}		–0.1	V
Output Current Source	$I_{IOUT(Source)}$		3	mA
Output Current Sink	$I_{IOUT(Sink)}$		10	mA
Overcurrent Transient Tolerance	I_P	1 pulse, 100 ms	100	A
Nominal Operating Ambient Temperature	T_A	Range E	–40 to 85	°C
Maximum Junction Temperature	T_J(max)		165	°C
Storage Temperature	T_{stg}		–65 to 170	°C

Y las características típicas de funcionamiento:

COMMON OPERATING CHARACTERISTICS [1]: Over full range of T_A, C_F = 1 nF, and V_{CC} = 5 V, unless otherwise specified

Characteristic	Symbol	Test Conditions	Min.	Typ.	Max.	Units
ELECTRICAL CHARACTERISTICS						
Supply Voltage	V_{CC}		4.5	5.0	5.5	V
Supply Current	I_{CC}	V_{CC} = 5.0 V, output open	–	10	13	mA
Output Capacitance Load	C_{LOAD}	VIOUT to GND	–	–	10	nF
Output Resistive Load	R_{LOAD}	VIOUT to GND	4.7	–	–	kΩ
Primary Conductor Resistance	$R_{PRIMARY}$	T_A = 25°C	–	1.2	–	mΩ
Rise Time	t_r	$I_P = I_P$(max), T_A = 25°C, C_{OUT} = open	–	3.5	–	µs
Frequency Bandwidth	f	–3 dB, T_A = 25°C; I_P is 10 A peak-to-peak	–	80	–	kHz
Nonlinearity	E_{LIN}	Over full range of I_P	–	1.5	–	%
Symmetry	E_{SYM}	Over full range of I_P	98	100	102	%
Zero Current Output Voltage	$V_{IOUT(Q)}$	Bidirectional; I_P = 0 A, T_A = 25°C	–	V_{CC} × 0.5	–	V
Power-On Time	t_{PO}	Output reaches 90% of steady-state level, T_J = 25°C, 20 A present on leadframe	–	35	–	µs
Magnetic Coupling [2]			–	12	–	G/A
Internal Filter Resistance [3]	$R_{F(INT)}$			1.7		kΩ

[1] Device may be operated at higher primary current levels, I_P, and ambient, T_A, and internal leadframe temperatures, T_A, provided that the Maximum Junction Temperature, T_J(max), is not exceeded.

[2] 1G = 0.1 mT.

[3] $R_{F(INT)}$ forms an RC circuit via the FILTER pin.

Para la versión de 30A se tiene la siguientes performance:

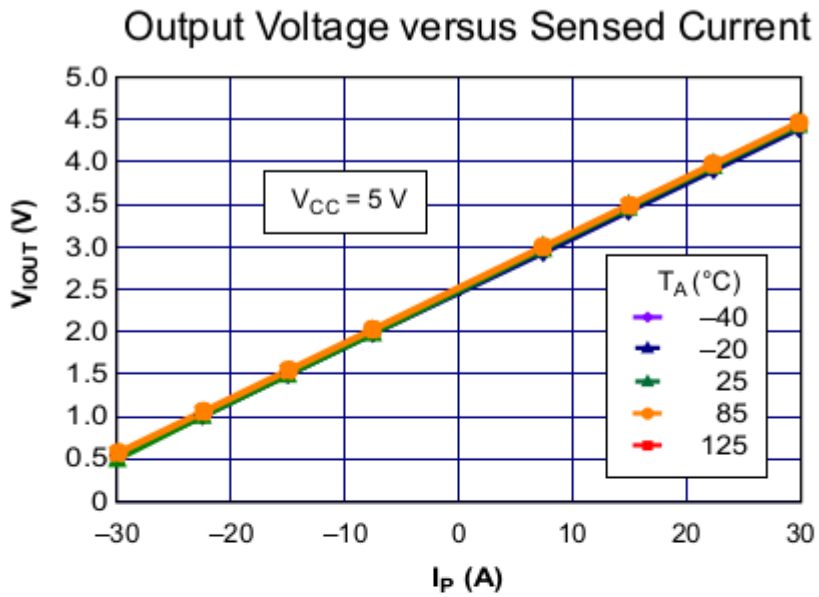
x30A PERFORMANCE CHARACTERISTICS [1]: T_A = –40°C to 85°C, C_F = 1 nF, and V_{CC} = 5 V, unless otherwise specified

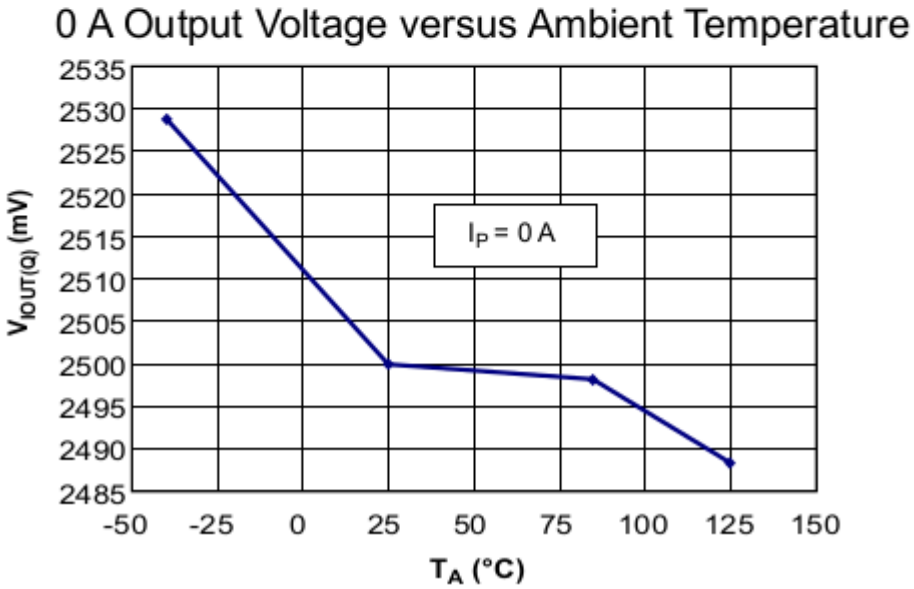
Characteristic	Symbol	Test Conditions	Min.	Typ.	Max.	Units
Optimized Accuracy Range	I_P		–30	–	30	A
Sensitivity	Sens	Over full range of I_P, T_A = 25°C	63	66	69	mV/A
Noise	$V_{NOISE(PP)}$	Peak-to-peak, T_A = 25°C, 66 mV/A programmed Sensitivity, C_F = 47 nF, C_{OUT} = open, 2 kHz bandwidth	–	7	–	mV
Zero Current Output Slope	$\Delta V_{OUT(Q)}$	T_A = –40°C to 25°C	–	–0.35	–	mV/°C
		T_A = 25°C to 150°C	–	–0.08	–	mV/°C
Sensitivity Slope	ΔSens	T_A = –40°C to 25°C	–	0.007	–	mV/A/°C
		T_A = 25°C to 150°C	–	–0.002	–	mV/A/°C
Total Output Error [2]	E_{TOT}	I_P = ±30 A, T_A = 25°C	–	±1.5	–	%

[1] Device may be operated at higher primary current levels, I_P, and ambient temperatures, T_A, provided that the Maximum Junction Temperature, T_J(max), is not exceeded.

[2] Percentage of I_P, with I_P = 30 A. Output filtered.

Se observa que posee buena linealidad dentro de los rangos normales de funcionamiento:





Módulo tipo Arduino:

El módulo en donde se utiliza el ACS712 se muestra a continuación:

El sensor nos entrega un valor de 2.5 v para una corriente de 0A y a partir de allí incrementa proporcionalmente de acuerdo a la sensibilidad, teniendo una relación lineal entre la salida de tensión del sensor y la corriente. Posee un ancho de banda de 50KHz.

Dicha relación es una línea recta en una gráfica Voltaje vs Corriente donde la pendiente es la sensibilidad y la intersección en el eje Y es 2.5 voltios. La ecuación de la recta seria la siguiente:

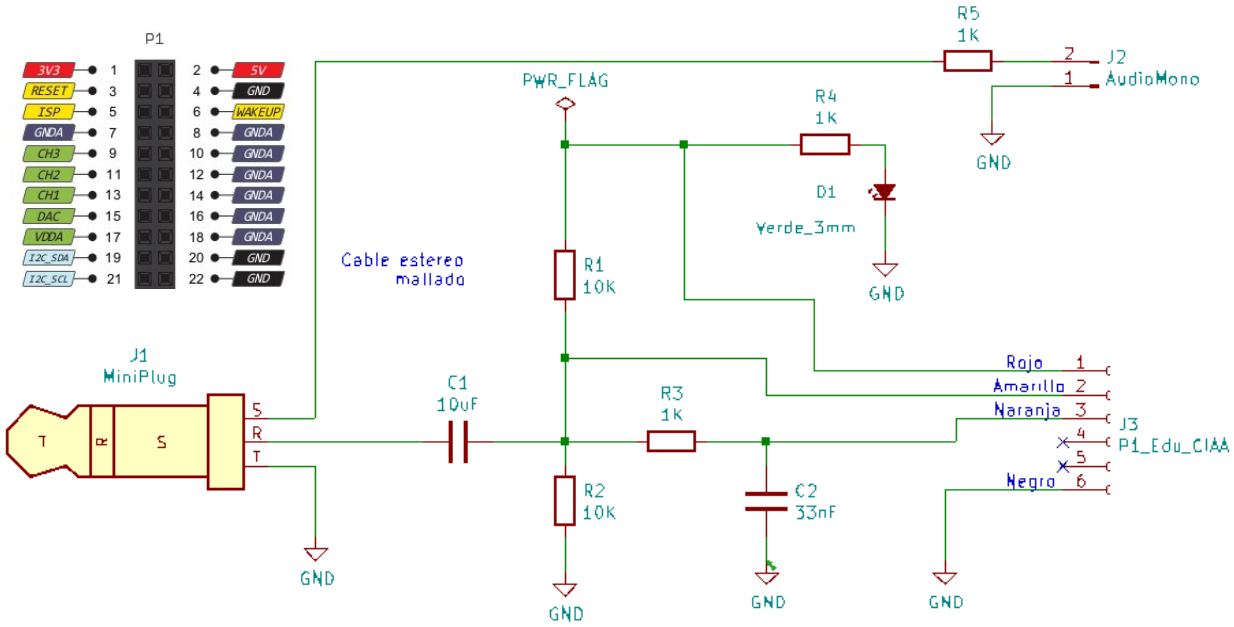
$$V = m\,I + 2.5$$

Donde la pendiente es m y equivale a la Sensibilidad. Despejando tendremos la ecuación para hallar la corriente a partir de la lectura del sensor:

$$I = \frac{V - 2.5}{Sensibilidad}$$

Circuito utilizado para pruebas(anti alias)

Se conectará en el port P1 de la EDU-CIAA el siguiente circuito:



Programa generador de señales usado para prueba

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.signal as sc
import simpleaudio as sa
import os


N=50000             # Cantidad de períodos
cantidad=2           # Cantidad de veces que se repite la señal
fase    = 0          # Fase de la señal en radianes
amp     = 1.0        # Ampitud de la señal
f       = 50         # Frecencia de la señal
fsec    = 2500       # Frecuencia de batido
amp2    = 0.1
fs      = 44100      # Frecuencia de muestreo en Hz, ver frecuencias soportadas de
                     # la placa de sonido


def stop_tx():
        if (sa.PlayObject.is_playing):
            sa.stop_all()


# Función senoidal
# Parámetros:
#       fs      --> fecuencia de sampleo
#       f       --> frecuencia de la señal de entrada
#       amp     --> amplitud del señal de 0 a 1.
#       muestras--> cantidad de veces que se repite la señal
#       fase    --> fase de la señal en radianes
# Devuelve:
#       f1      --> vector de señal de la señal
#       n       --> vector de tienpos de sampling
def senoidal(fs,f,amp,muestras,fase):
    n = np.arange(0, muestras/f, 1/fs)        # Intervalo de tiempo en segundos
    f1=(2**15-1)*amp*np.sin(2*np.pi*f*n+fase)         # Definimos el Vector de Frecuencias
    return f1,n

# Función cuadrada
# Parámetros:
#       fs      --> fecuencia de sampleo
#       f       --> frecuencia de la señal de entrada
#       amp     --> amplitud del señal de 0 a 1.
#       muestras--> cantidad de veces que se repite la señal
# Devuelve:
#       t       --> vector de valores temporales
#       senal   --> vector de valores de la señal
def cuadrada(fs,f,amp,muestras):
    n = np.arange(0, muestras/f, 1/fs)        # Intervalo de tiempo en segundos
    return (2**15-1)*amp*sc.square(2*np.pi*n*f),n



# Función triangular
# Parámetros:
#       fs      --> fecuencia de sampleo
#       f       --> frecuencia de la señal de entrada
#       amp     --> amplitud del señal de 0 a 1.
#       muestras--> cantidad de veces que se repite la señal
# Devuelve:
#       t       --> vector de valores temporales
#       senal   --> vector de valores de la señal
def triangular(fs,f,amp,muestras):
    n = np.arange(0, muestras/f, 1/fs)        # Intervalo de tiempo en segundos
    return (2**15-1)*sc.sawtooth(2*np.pi*f*n,1),n
```

```
def senoidalSuma(fs,f,amp,muestras,fase,B,amp2):
    n = np.arange(0, muestras/f, 1/fs)        # Intervalo de tiempo en segundos
      # Definimos el Vector de Frecuencias
    f1=(2**15-1)*amp*np.sin(2*np.pi*f*n+fase)+(2**15-1)*amp2*np.sin(2*np.pi*B*n)
    return f1,n

def senoidalB(fs,f,amp,muestras,fase,B):
    n = np.arange(0, muestras/f, 1/fs)        # Intervalo de tiempo en segundos
    f1=(2**15-1)*np.sin(2*np.pi*B/2*n*n)      #sweept
    return f1,n

# Grafica la señal
def graficar(encabezado,funcion,n,xlim):
    global f
    fig = plt.figure(1)
    plt.suptitle(encabezado)
    plt.subplots_adjust(left=0.08, bottom=0.08, right=0.98, top=0.9, wspace=0.4, hspace=0.8)

    s1 = fig.add_subplot(1,1,1)
    plt.title("Señal")
    plt.xlabel("Tiempo(s)")
    plt.ylabel("Amplitud")
    plt.xlim(0,xlim)
    s1.grid(True)
    s1.plot(n,funcion*1.65/(2**15-1),'b-')
    plt.show()
    return


def reproducir(note):
    audio = note.astype(np.int16)                 #tranforma la variable note a entero de 16bits y lo guarda en audio
    for i in range(cantidad):
        play_obj = sa.play_buffer(audio, 1, 2, fs)  # sale el audio
       # play_obj.wait_done()                      # espera que termine la linea anterior


def op_senoidal():
    global f,amp,N,fs,fase
    os.system("clear")
    print("==============================")
    print("=======Señal Senoidal=========")
    print("==============================")
    print("La frecuencia de la señal a reproducir es ={}Hz".format(f))
    print("La amplitud de la señal de red \t\tamp={}v-->{}%".format(amp*1.65,amp*100))
    f1,n=senoidal(fs,f,amp,N,fase)
    consulta=input("Desea graficar la señal S o N[Enter] :")
    if consulta=='S' or consulta =='s':
        encabezado="Senoidal -->"+" f="+str(f)+"Hz"+" T="+str((1/f)*1000)+"mseg"+"  N="+str(N)+"  fs="+str(fs)+"Hz"+"
fase="+str(fase*180/np.pi)+"º"
        graficar(encabezado,f1,n,2/f)
    consulta=input("Desea reproducir la señal S o N[Enter] :")
    if consulta=='S' or consulta =='s':
        reproducir(f1)
    return

def op_ruido():
    global fsec,amp2,N,fs
    os.system("clear")
    print("==============================")
    print("=======Señal Senoidal=========")
    print("==============================")
    print("La frecuencia de la señal a reproducir es ={}Hz".format(fsec))
    print("La amplitud de la señal de red \t\tamp2={}v-->{}%".format(amp2*1.65,amp2*100))
    f1,n=senoidal(fs,fsec,amp2,N,0)
    consulta=input("Desea graficar la señal S o N[Enter] :")
    if consulta=='S' or consulta =='s':
        encabezado="Senoidal -->"+" f="+str(fsec)+"Hz"+" T="+str((1/fsec)*1000)+"mseg"+"  N="+str(N)+"  fs="+str(fs)+"Hz"+"
fase="+str(fase*180/np.pi)+"º"
```

```
        graficar(encabezado,f1,n,2/fsec)
    consulta=input("Desea reproducir la señal S o N[Enter] :")
    if consulta=='S' or consulta =='s':
        reproducir(f1)
    return

def op_cuadrada():
    global f
    os.system("clear")
    print("==============================")
    print("=======Señal Cuadrada=========")
    print("==============================")
    print("La frecuencia de la señal a reproducir es ={}Hz".format(f))
    print("La amplitud de la señal de red \t\tamp={}v-->{}%".format(amp*1.65,amp*100))
    f1,n=cuadrada(fs,f,amp,N)
    consulta=input("Desea graficar la señal S o N[Enter] :")
    if consulta=='S' or consulta =='s':
        encabezado="Cuadrada -->"+" f="+str(f)+"Hz"+" T="+str((1/f)*1000)+"mseg"+"  N="+str(N)+"  fs="+str(fs)+"Hz"+"
fase="+str(fase*180/np.pi)+"°"
        graficar(encabezado,f1,n,2/f)
    consulta=input("Desea reproducir la señal S o N[Enter] :")
    if consulta=='S' or consulta =='s':
        reproducir(f1)
    return

def op_triangular():
    global f
    os.system("clear")
    print("==============================")
    print("======Señal Triangular========")
    print("==============================")
    print("La frecuencia de la señal a reproducir es ={}Hz".format(f))
    print("La amplitud de la señal de red \t\tamp={}v-->{}%".format(amp*1.65,amp*100))
    f1,n=triangular(fs,f,amp,N)
    consulta=input("Desea graficar la señal S o N[Enter] :")
    if consulta=='S' or consulta =='s':
        encabezado="Triangular -->"+" f="+str(f)+"Hz"+" T="+str((1/f)*1000)+"mseg"+"  N="+str(N)+"  fs="+str(fs)+"Hz"+"
fase="+str(fase*180/np.pi)+"°"
        graficar(encabezado,f1,n,2/f)
    consulta=input("Desea reproducir la señal S o N[Enter] :")
    if consulta=='S' or consulta =='s':
        reproducir(f1)
    return

def op_senoidalB():
    global B
    os.system("clear")
    print("==============================")
    print("=Señal Barrido de Senoidales==")
    print("==============================")
    print("La frecuencia de la señal a reproducir es ={}Hz".format(f))
    print("La amplitud de la señal de red \t\tamp={}v-->{}%".format(amp*1.65,amp*100))
    f1,n=senoidalB(fs,f,amp,N,fase,B)
    consulta=input("Desea graficar la señal S o N[Enter] :")
    if consulta=='S' or consulta =='s':
        encabezado="Barrido de 2 senoidales-->"+" f="+str(f)+"Hz"+" T="+str((1/f)*1000)+"mseg"+"  N="+str(N)+"  fs="+str(fs)
+"Hz"+"  fase="+str(fase*180/np.pi)+"°"
        graficar(encabezado,f1,n,200/f)
    consulta=input("Desea reproducir la señal S o N[Enter] :")
    if consulta=='S' or consulta =='s':
        reproducir(f1)
    return


def op_senoidalSuma():
    global f
    os.system("clear")
    print("==============================")
    print("===Señal Suma de Senoidales===")
    print("==============================")
```

```
    print("La frecuencia de la señal a reproducir es ={}Hz".format(f))
    print("La amplitud de la señal es={}".format(amp))
    f1,n=senoidalSuma(fs,f,amp,N,fase,fsec,amp2)
    consulta=input("Desea graficar la señal S o N[Enter] :")
    if consulta=='S' or consulta =='s':
        encabezado="Suma de 2 senoidales -->"+" f="+str(f)+"Hz"+" T="+str((1/f)*1000)+"mseg"+"  N="+str(N)+"  fs="+str(fs)
+"Hz"+"  fase="+str(fase*180/np.pi)+"º"
        graficar(encabezado,f1,n,2/f)
    consulta=input("Desea reproducir la señal S o N[Enter] :")
    if consulta=='S' or consulta =='s':
        reproducir(f1)
    return

def valores():
    global N,fs,amp,fase,cantidad,fsec,f,amp2
    os.system("clear")
    print("Los valores actuales son:")
    print("----------------------------------------------------------------------")
    print("La frecuencia de sampling \t\tfs={}Hz --> \tts={:.4f}mseg".format(fs,1/fs*1000))
    print("La cantidad de períodos es \t\tN={}".format(N))
    print("La cantidad de veces a repetir es\tN1={}".format(cantidad))
    print("----------------------------------------------------------------------")
    print("La frecuencia de red \t\t\tfred={} Hz".format(f))
    print("La amplitud de la señal de red \t\tamp={}v-->{}%".format(amp*1.65,amp*100))
    print("La fase de la señal es\t\t\tfase={}*Pi radianes".format(fase))
    print("----------------------------------------------------------------------")
    print("La frecuencia de ruido es \t\tfr={}Hz".format(fsec))
    print("La amplitud de la señal de ruido \tamp2={}v-->{}%".format(amp2*1.65,amp2*100))
    print("----------------------------------------------------------------------")
    consulta=input("Desea cambiar los valores S o N[Enter] :")
    if consulta=='S' or consulta =='s':
        valor=input("Ingrese la frecuencia Hz de sampling \t\t\tfs=")
        if valor.isdigit():
            fs=int(valor)
            print("\t\tEl tiempo de sampleo es ts={:.4f}mseg".format(1/fs*1000))
        valor=input("Ingrese la cantidad de períodos a visualzar \t\tN=")
        if valor.isdigit():
            N=int(valor)
        valor=input("Ingrese la cantidad de veces que desea repetir la señal visualizada =")
        if valor.isdigit():
            cantidad=int(valor)
        valor=input("Ingrese la frecuencia de red en Hz de la señal \t\tfred=")
        if valor.isdigit():
            f=int(valor)
        valor=input("Ingrese la amplitud de la señal de red(0-100)% \t\tamp=")
        if valor.isdigit():
            if(int (valor)>100):
                valor=100
            amp=float(valor)/100
        valor=input("Ingrese la fase en grados(enteros) entre 0 a 360 \tfase=")
        if valor.isdigit():
            fase=float(valor)*np.pi/180
        valor=input("Ingrese la frecuencia de ruido en Hz de  \t\tfr=")
        if valor.isdigit():
            fsec=int(valor)
        valor=input("Ingrese la amplitud de la señal de ruido(0-100)%\tamp2=")
        if valor.isdigit():
            if(int (valor)>100):
                valor=100
            amp2=float(valor)/100
        input("Presiona cualquier tecla para continuar")
        os.system("clear")


#============================================================
# Inicio del programa principal
#============================================================
menu="""
Programas de la transformada Discreta de Fourier
elija una opción:
```

```
[1] Señal de red senoidal
[2] Señal de ruido senoidal
[3] Señal de red cuadrada
[4] Señal de red triangular
[5] Suma de frecuencias senoidales (red+ruido)amp*fred+amp2*fr
[6] Barrido de frecuencias senoidales de fred a fr

[7] Termina Tx de sonido
[8] Seteo de frecuencia de la señal de entrada, número de muestras,
    frecuencia de sampling.
[9] Salir
"""

while(True):
    os.system("clear")
    print(menu)

    opcion=input("Elija una opción: ")

    if opcion== '1':
        op_senoidal()
    elif opcion== '2':
        op_ruido()
    elif opcion== '3':
        op_cuadrada()
    elif opcion== '4':
        op_triangular()
    elif opcion== '5':
        op_senoidalSuma()
    elif opcion== '6':
        op_senoidalB()
    elif opcion== '7':
        stop_tx()
    elif opcion== '8':
        valores()
    elif opcion== '9':
        os.system("clear")
        print("Gracias por usar el programa !!!")
        exit (0)
    else:
        print("No selecionó una opción válida\n\r")
```

Programa visualizador de los datos enviados por la EDU-CIAA

```
#!python3
import numpy as np
import matplotlib.pyplot as plt
from   matplotlib.animation import FuncAnimation
import os
import io
import serial

FFT_MAX_DEFAULT=0.1
Vp_MAX=1.65

STREAM_FILE=("/dev/ttyUSB1","serial")
#STREAM_FILE=("log.bin","file")

header = { "pre": b"*header*", "id": 0, "N": 256, "fs": 10000,
"cutFrec":0,"cutFrec2":0,"ruido":0,"M":10,"pos":b"end*" }
fig    = plt.figure ( 1 )
fig.suptitle('Ruido en la red de 220v con CIAA', fontsize=16)

#--------------------------ADC-------------------------------------------------------------
adcAxe = fig.add_subplot ( 3,1,1 )
adcLn, = plt.plot        ( [],[],'r-o',linewidth=12, alpha = 0.3 ,label  = "adc")
adcAxe.grid              ( True )
adcAxe.set_ylim          ( -Vp_MAX ,Vp_MAX )

#----------------------ciaaFFT vs fft(adc)-----------------------------------------------
fftAxe       = fig.add_subplot ( 3,1,2 )
fftAxe.clear()
fftAxe.set_ylim ( 0,FFT_MAX_DEFAULT)      #np.max(absFft))  ---> 0.03
#cutFrecZoneLn   = fftAxe.fill_between([0,0],100,-100,facecolor = "yellow",alpha = 0.2)
fftAxe.set_title("fft(ciaaFFT) vs fft(adc)",rotation = 0,fontsize = 10,va = "center")
fftLn,     = plt.plot ( [] ,[] ,'r-o' ,linewidth = 10  ,alpha = 0.3 ,label = "abs(FFT(adc))" )
ciaaFftLn, = plt.plot ( [] ,[] ,'b-o' ,linewidth = 3 ,alpha = 0.8 ,label = "ciaaFFT filtered out" )
fftLg      = fftAxe.legend()
#fftAxe.text([],[],"Ampl. ruido="+str(f'{fftAbsMax*1.65:.{4}f}')+"v",fontsize=20)

#----------------------------Valores---------------------------------------------------------
valoresRx       = fig.add_subplot ( 3,1,3 )
valoresRx.clear()
valoresRx.set_ylim ( 0,10)
valoresRx.set_xlim ( 0,10)
plt.axis('off')
valoresRx.spines['right'].set_visible(False)
valoresRx.spines['top'].set_visible(False)
valoresRx.spines['bottom'].set_visible(False)
valoresRx.spines['left'].set_visible(False)
N_Rx=valoresRx.text(0,8,' ',fontsize=10)
M_Rx=valoresRx.text(0,6,' ',fontsize=10)
fs_Rx=valoresRx.text(0,4,' ',fontsize=10)
cutFrec_Rx=valoresRx.text(0,2,' ',fontsize=10)
cutFrec2_Rx=valoresRx.text(0,0,' ',fontsize=10)
ruido_Rx=valoresRx.text(5,8,' ',fontsize=10)
muestras=valoresRx.text(5,6,' ',fontsize=10)
vp_pos_RX=valoresRx.text(5,4,' ',fontsize=10)
vp_neg_RX=valoresRx.text(5,2,' ',fontsize=10)
mod_FFT_Rx=valoresRx.text(5,0,' ',fontsize=10)
```

```
def findHeader(f,h):
    data=bytearray(b'12345678')
    while data!=h["pre"]:
        data+=f.read(1)
        if len(data)>len(h["pre"]):
            del data[0]
    h["id"]      = readInt4File(f,4)
    h["N" ]      = readInt4File(f)
    h["fs"]      = readInt4File(f)
    h["cutFrec"] = readInt4File(f)
    h["cutFrec2"] = readInt4File(f)
    h["ruido"]   = readInt4File(f)
    h["M"]       = readInt4File(f)
    data=bytearray(b'1234')
    while data!=h["pos"]:
        data+=f.read(1)
        if len(data)>len(h["pos"]):
            del data[0]
    print({k:round(v,2) if isinstance(v,float) else v for k,v in h.items()})
    return h["id"],h["N"],h["fs"],h["cutFrec"],h["cutFrec2"],h["ruido"],h["M"]

def readInt4File(f,size=2,sign=False):
    raw=f.read(1)
    while( len(raw) < size):
        raw+=f.read(1)
    return (int.from_bytes(raw,"little",signed=sign))

def flushStream(f,h):
    if(STREAM_FILE[1]=="serial"): #pregunto si estoy usando la bibioteca pyserial o un file
        f.flushInput()
    else:
        f.seek ( 2*(h["N"]+h["M"]-1),io.SEEK_END)

def readSamples(adc,synth,N,trigger=False,th=0):
    state="waitLow" if trigger else "sampling"
    i=0
    for t in range(N):
        sample = (readInt4File(streamFile,sign = True)*1.65)/(2**6*512)
        ciaaFFT = (readInt4File(streamFile,sign = True)*1.65)/(2**1*512)
        state,nextI= {
              "waitLow" : lambda sample,i: ("waitHigh",0) if sample<th else ("waitLow" ,0),
              "waitHigh": lambda sample,i: ("sampling",0) if sample>th else ("waitHigh",0),
              "sampling": lambda sample,i: ("sampling",i+1)
              }[state](sample,i)
        adc[i]=sample
        synth[i]=ciaaFFT
        i=nextI

def update(t):
    global header
    flushStream ( streamFile,header )
    id,N,fs,cutFrec,cutFrec2,ruido,M=findHeader ( streamFile,header )
    nData     = np.arange(0,N+M-1,1) #arranco con numeros enteros para evitar errores de float
    adc       = np.zeros(N+M-1)
    ciaaFFT = np.zeros(N+M-1).astype(complex)
    tData     = nData/fs

    # Leo los datos recibidos
    readSamples(adc,ciaaFFT,N+M-1,False,0)

    #==================================#=
    # Gráfico en función de la muestras                    #=
```

```
    adcAxe.set_xlim ( 0    ,(N+M-1) )                    #=
    adcLn.set_data  ( nData ,adc  )                      #=
    #====================================#=

   #=============================================================================================================#=
    #                          Gráfico del abs(FFT(adc)) y  del la señal filtrada                                  #=
    # if(fftAbsMax<FFT_MAX_DEFAULT):
    #     fftAbsMax=FFT_MAX_DEFAULT
    # fftAxe.set_ylim ( 0, fftAbsMax)              #Escalo al máximo

    #fftAxe.clear()

    fftAxe.grid     ( True   )

    fftAxe.set_xlim ( -fs/2,fs/2-fs/(N+M-1))
    fData=nData*fs/(N+M-1)-fs/2
    # Datos del gáfico
    ciaaFftLn.set_data (fData ,np.fft.fftshift(ciaaFFT))
    fftLn.set_data (fData ,np.abs(np.fft.fftshift(np.fft.fft(adc))/(N+M-1))**2)

    #auto escala el eje y, pero no tan bajo
    fftAxe.set_ylim ( 0,np.clip(np.max(ciaaFFT),0.1,300))
    #fftAxe.set_ylim ( 0,np.max(ciaaFFT))
    #fftAxe.set_ylim ( 0,np.max(np.fft.fft(adc)))

    cutFrecZoneLn = fftAxe.fill_between([-cutFrec,-cutFrec2],300,-300,facecolor="yellow",alpha=0.5)
    cutFrecZoneLn = fftAxe.fill_between([cutFrec2,cutFrec],300,-300,facecolor="yellow",alpha=0.5)



    #=============================================================================================================#=

    #=============================================================================================================#=
    #                  Calculo el fft en la PC con los valores recibidos en adc                                   #=
    muestras.set_text("Muestras="+str(id))
    N_Rx.set_text("Cant de muestras del adc N="+str(N))
    M_Rx.set_text("Cant de muestras del filtro N="+str(M))
    fs_Rx.set_text("fs="+str(fs)+"Hz")
    cutFrec_Rx.set_text("Frec. de corte superior="+str(cutFrec)+"Hz")
    cutFrec2_Rx.set_text("Frec. de corte inferior="+str(cutFrec2)+"Hz")
    ruido_Rx.set_text("Ruido="+str(ruido))
    vp_pos=0
    vp_neg=0
    for i in range(0,len(adc)):
        if(vp_pos<adc[i]):
            vp_pos=adc[i]
        if(vp_neg>adc[i]):
            vp_neg=adc[i]
    vp_pos_RX.set_text("Vp+="+str(f'{vp_pos:.{4}f}')+"v")
    vp_neg_RX.set_text("Vp-="+str(f'{vp_neg:.{4}f}')+"v")
    mod_FFT_Rx.set_text("|ciaaFFT|max="+str(f'{np.abs(np.max(ciaaFFT)):.{4}f}'))
    #=============================================================================================================#=


    return adcLn, ciaaFftLn, fftLn,
cutFrecZoneLn,muestras,N_Rx,M_Rx,fs_Rx,cutFrec_Rx,cutFrec2_Rx,ruido_Rx,vp_pos_RX,vp_neg_RX,mod_FFT_R
x

#seleccionar si usar la biblioteca pyserial o leer desde un archivo log.bin
if(STREAM_FILE[1]=="serial"):
    streamFile = serial.Serial(port=STREAM_FILE[0],baudrate=460800,timeout=None)
else:
    streamFile=open(STREAM_FILE[0],"rb",0)
```

```
ani=FuncAnimation(fig,update,10000,init_func=None,blit=True,interval=100,repeat=True)
plt.get_current_fig_manager().window.showMaximized() #para QT5
plt.show()
streamFile.close()
```