# Proyecto1

Laboratorio 3

## Conquista de gnuradio a nivel de programación

## Prácticas de programación con Python y GNU Radio

Lab3

**Autores:** Alex Julian Mantilla Rios

**Perteneciente al grupo:** B1A.G6

## Aspectos a mejorar en la guia

Los siguientes son apuntes del profesor para introducir mejoras a futuras prácticas:

- Por ahora no hay apuntes

## El Problema:

Por ahora el problema a resolver consiste en que el estudiante tiene suficientes bases de programación por objetos en Python pero no tiene experiencias con el uso de la libreria GNU Radio y el paradigma de programación que hay detrás de ella.

## El objetivo general es:

Usar la librería GNU Radio combinada con programación por objetos en Python para probar un ejemplo propuesto y luego poder implementar soluciones propias que ojalá se orienten a la generación y visualización señales aletorias en tiempo y frecuencia.

## Preparativos

- Baje una version actualizada del: el libro de la asignatura. Observe que en los capítulos del libro ofrecen enlaces a códido de software, a flujogramas y otros recursos que son parte del libro. Por ejemplo, observa que debajo de cada gráfica con flujogramas hay una nota que dice: “Flujograma usado”. Esos recursos usados en el libro están en la página del libro:  https://sites.google.com/saber.uis.edu.co/comdig/sw

## Apuntes sobre el uso de la libreria GNURadio para nunca olvidar

- Las ayudas sobre este tema se encuentran en [5], punto 2.3
- observe que todo bloque de gnuradio tiene un nombre que termina con una o dos letras así:
    - nombre_f: significa que el bloque solo tiene entradas o solo salidas y que son de tipo flotante
    - nombre_ff: el bloque tiene entradas y salidas y ambas son de tipo flotante
    - nombre_c: el bloque solo tiene entradas o solo salidas y que son de tipo complejo
    - nombre_cc: el bloque tiene entradas y salidas y ambas son de tipo
    - nombre_fc: el bloque tiene entradas y salidas y las entradas son de tipo flotante y las salidas son de tipo complejo.
    - [faltan mas casos]

## Objetivos específicos

- Implemente y corra el ejemplo propuesto en [5], capitulo 2.3 . Lo realmente importante es comprender cómo es que se programa en gnuradio.
    a. en qué consiste el paradigma de programación.
    b. describir lo que hace la solución.
- Realice modificaciones al código que le permitan comprobar que comprende correctamente el código:
    a. haga cambios en los nombres de funciones, pruebe usar otros valores para los parámetros.
    b. agregue un sistema de visualización gracias a que tiene otros ejemplos en [5]
- Aprenda a usar la documentación de GNURadio. Siga las indicaciones de la wiki de gnuradio [6], capítulo 3.1.3.
    a. Debe buscar y entender la documentación de cada uno de los bloques usados en los puntos anteriores. Debe usar la documentación para encontrar nuevos bloques y poder implementar una solución más completa orientada a generar señales aleatorias y observarlas en tiempo y frecuencia

## Referencias usadas


[1] Manual de manuales
[2] El libro de la asignatura
[3] Página del libro
[4] Libro Internet de los Objetos
[5] Manual de programación en código de python y GNURadio
[6] wiki de GNURadio

## Informe de resultados

Desarrollo del Objetivo 1. Presente a continuación los resultados del objetivo 1.

- **¿En qué consiste el paradigma de programación?** El paradigma es utilizar la programación orientada a objetos para definir todo el sistema de SDR, es decir, es un intento por hacer una analogía de los diagramas de bloques teóricos. De esta forma se define una clase “top_block” que es el sistema completo (permitiendo diferentes instanciaciones del mismo bloque). Dentro de esta clase “top_block” se instanciarán objetos de la clase “block” que corresponden a los bloques de procesos del sistema, con sus respectivas entradas y salidas, como se representaría en el diagrama de bloques teórico.
  Todo esto, al ser software, si se tiene la sufiente capacidad de cómputo, puede acercase a un sistema en tiempo real, tal como lo haría un sistema físico.

- **Describir lo que hace la solución.** La solución se muestra a continuación:

```
from gnuradio import gr
from gnuradio import audio
from gnuradio import analog

########################################################
# LA CLASE QUE DESCRIBE TODO EL FLUJOGRAMA
########################################################
class my_top_block(gr.top_block):        # hereda de gr.top_block
    def __init__(self):
        gr.top_block.__init__(self)     # otra vez la herencia

        sample_rate = 32000
        ampl = 0.1

        src0 = analog.sig_source_f(sample_rate, analog.GR_SIN_WAVE, 350, ampl)
        src1 = analog.sig_source_f(sample_rate, analog.GR_SIN_WAVE, 440, ampl)
        dst = audio.sink(sample_rate, "")
        self.connect(src0, (dst, 0))
        self.connect(src1, (dst, 1))
########################################################
# EL CÓDIGO PARA LLAMAR EL FLUJOGRAMA “my_top_block”
########################################################
my_top_block().run()
```

Aquí se está definiendo un sistema (o diagrama de bloques, “TOP”) llamado “my_top_block”, al instanciar la clase “my_top_block” que hereda de la clase “gr.top_block”. Dentro del constructor de esta clase, a parte de ejecutar el contructor de clase madre, se definen los parámetros “sample_rate” (o frecuencia de muestreo)

con valor de 32000 y “ampl” (o amplitud) con valor de 0.1 y se instancian los bloques que van a conformar este sistema. Estos bloques son nombrados “src0”, “src1” y “dst”:

a. **“src0”** Es un bloque que genera una señal con valores flotantes. Esta es una instancia de la clase “analog.sig_source_f”, y, con los parámetros sumistrados para el constructor, se está generando una señal senoidal de amplitud 0.1 y frecuencia 350 Hz, con una frecuencia de muestreo de 32 kSa.
b. **“src1”** Es un bloque que genera una señal con valores flotantes. Esta es una instancia de la clase “analog.sig_source_f”, y, con los parámetros sumistrados para el constructor, se está generando una señal senoidal de amplitud 0.1 y frecuencia 440 Hz, con una frecuencia de muestreo de 32 kSa.
c. **“dst”** Es un bloque que reproduce una señal en el parlante del computador. A este bloque se le está asignando como entrada las salidas de los bloques “src0” y “src1”.

De esta forma, al llamar “my_top_block().run()” se inicia la simulación del diagrama de bloques completo descrito en la clase “my_top_block”. A este diagrama de bloques general se le llama **Flujograma**.

Desarrollo del Objetivo 2. Presente a continuación los resultados del objetivo 2.

- **Haga cambios en los nombres de funciones, pruebe usar otros valores para los parámetros.** Los cambios más sencillos aquí pueden ser solo variar los parámetros definidos:
    a. Siendo **32000** el valor por defecto de **“sample_rate”,** puede ser cambiado a un valor más alto como **64000** para tener una mayor resolución de la señal, a cambio claro está, de mayor necesidad de cómputo.
    b. Además, cambiar el valor de **“ampl”** de **0.1** a **1** generará una señal senoidal de mayor amplitud, que se traduce en que el sonido reproducido será más fuerte.
    c. También, fijándonos en los parámetros del constructor para los bloques **“src0”** y **“src1”** en las líneas:

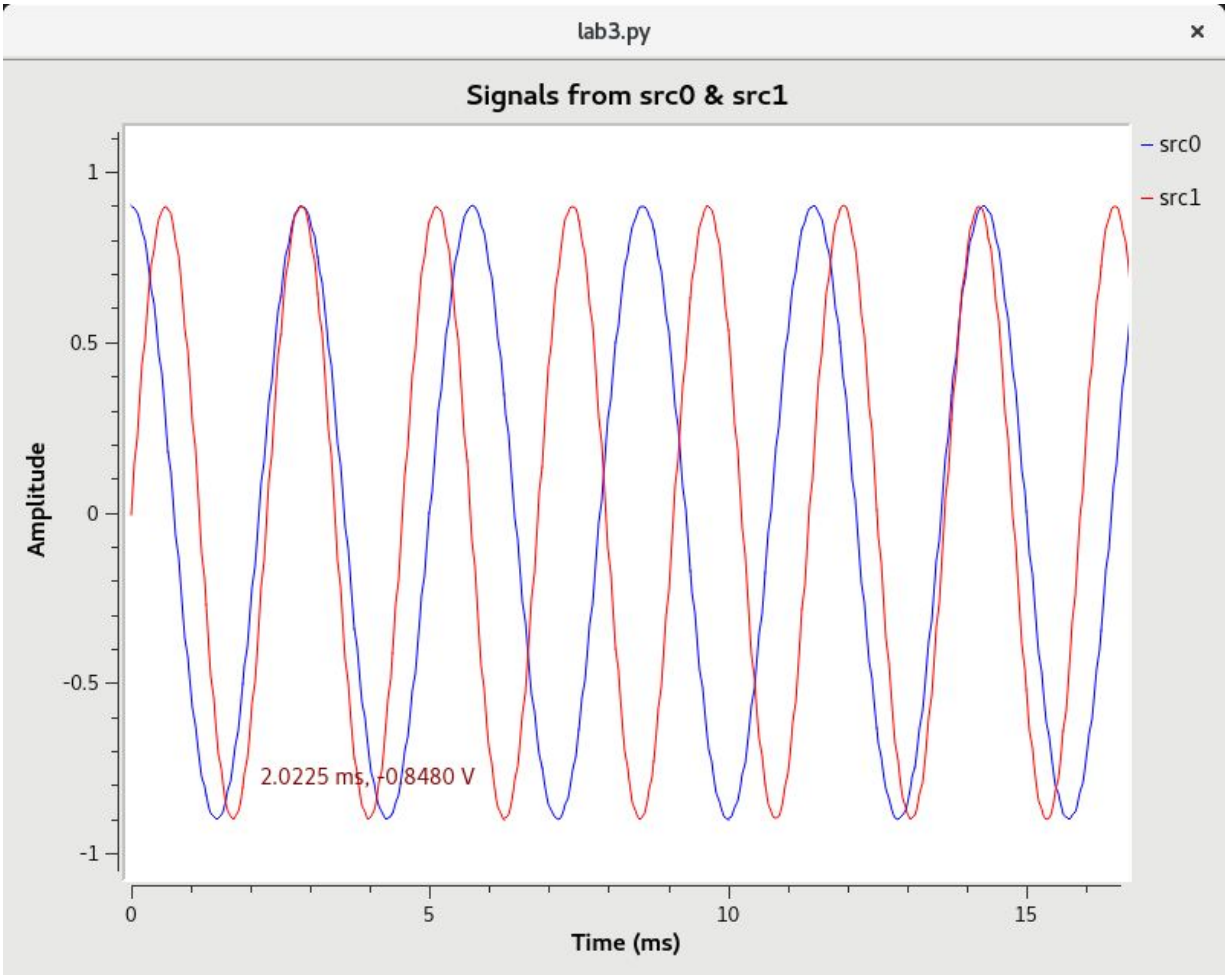
```
src0 = analog.sig_source_f(sample_rate, analog.GR_SIN_WAVE, 350, ampl)
src1 = analog.sig_source_f(sample_rate, analog.GR_SIN_WAVE, 440, ampl)
```

Podemos incluso cambiar la frecuencia de las señales generadas, por ejemplo para **“src0”** cambiar de **350 Hz** a **600 Hz** para escuchar un sonido más agudo.

- **Agregue un sistema de visualización gracias a que tiene otros ejemplos en [5].** Tomando como ejemplo [5] e investigando un poco más. Se llega al siguiente código para generar una visualización de lo que está ocurriendo:

```
from gnuradio import analog
# Librerias para poder incluir graficas tipo QT
from gnuradio import qtgui
from PyQt4 import Qt # si no se acepta PyQt4 cambie PyQt4 por PyQt5
import sys, sip
#####################################################
# LA CLASE QUE DESCRIBE TODO EL FLUJOGRAMA
#####################################################
class my_top_block(gr.top_block):        # hereda de gr.top_block
    def __init__(self):
        gr.top_block.__init__(self)     # otra vez la herencia
        ampl = 0.9
        size = 10000
        sample_rate = 32000
        plotName = "Signals from src0 & src1"
        nconnections = 2
        src0 = analog.sig_source_f(sample_rate, analog.GR_SIN_WAVE, 350, ampl)
        src1 = analog.sig_source_f(sample_rate, analog.GR_SIN_WAVE, 440, ampl)
        dst = audio.sink(sample_rate, "")
        self.snk = qtgui.time_sink_f(size,sample_rate,plotName,nconnections)
        self.connect(src0, (dst, 0))
        self.connect(src1, (dst, 1))
        self.connect(src0, (self.snk, 0))
        self.connect(src1, (self.snk, 1))
        pyWin = sip.wrapinstance(self.snk.pyqwidget(), Qt.QWidget)
        pyWin.show()
#####################################################
# EL CÓDIGO PARA LLAMAR EL FLUJOGRAMA "my_top_block"
#####################################################
def main():
    qapp = Qt.QApplication(sys.argv)
    myTop = my_top_block()
    myTop.start()
    qapp.exec_()
    myTop.stop()
if __name__ == "__main__":
    main()
```

En este código se implementa un osciloscopio virtual para observar las señales generadas por "src0" y "src1", usando una instanción de la clase "qtgui.time_sink_f". El resultado es el siguiente:




Donde la lína azul corresponde a la señal senoidal generada por el bloque **“src0”** y la lína azul corresponde a la señal senoidal generada por el bloque **“src1”**.

Desarrollo del Objetivo 3. Presente a continuación los resultados del objetivo 3.

- **Debe buscar y entender la documentación de cada uno de los bloques usados en los puntos anteriores. Debe usar la documentación para encontrar nuevos bloques y poder implementar una solución más completa orientada a generar señales aleatorias y observarlas en tiempo y frecuencia.**

  Los bloques de generación de señales “src” (analog.sig_source_f) no solamente pueden generar una señal senoidal, sino que pueden general diferentes tipos según el 2do parámetro del constructor (que en el ejemplo es analog.GR_SIN_WAVE), de la siguiente forma:

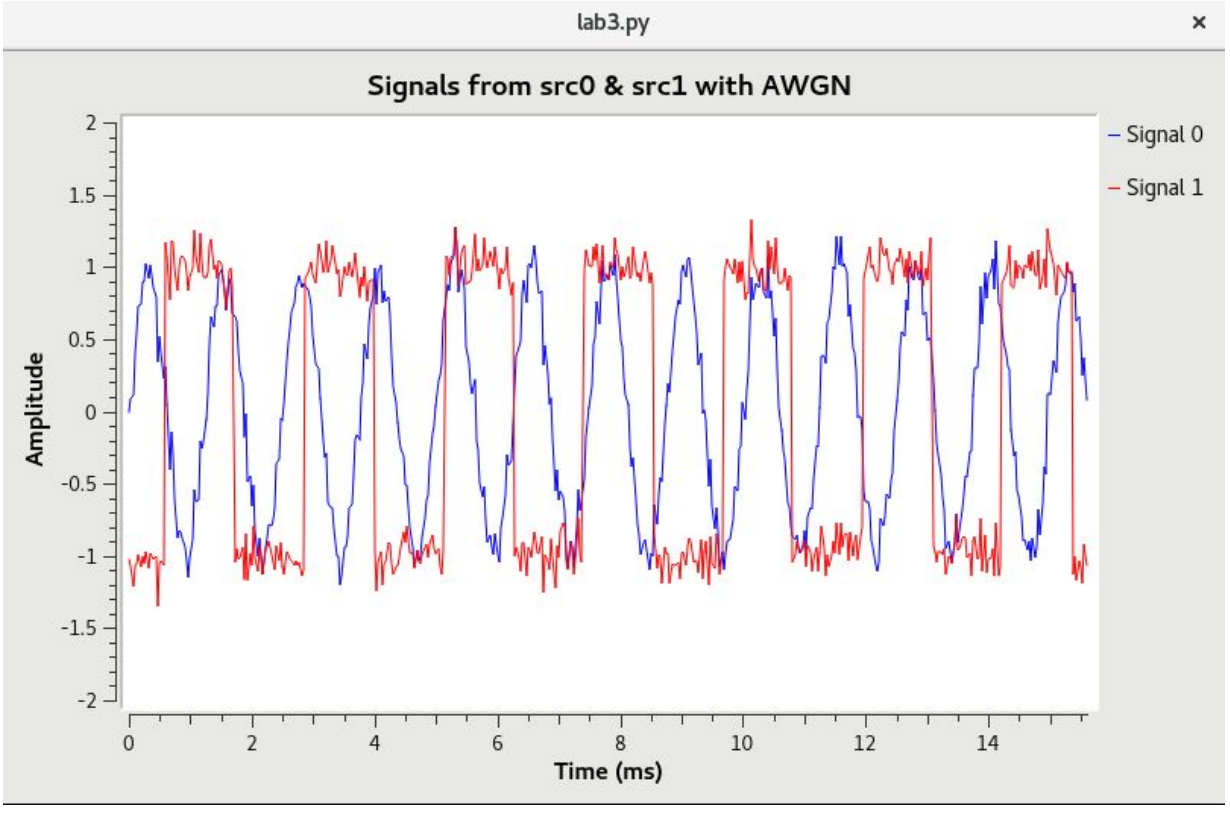
| Señal a generar | Referencia a usar |
|---|---|
| Constante | GR_CONST_WAVE |
| Senoidal | GR_SIN_WAVE |
| Cosenoidal | GR_COS_WAVE |
| Rectangular | GR_SQR_WAVE |
| Triangular | GR_TRI_WAVE |
| Diente de cierra | GR_SAW_WAVE |

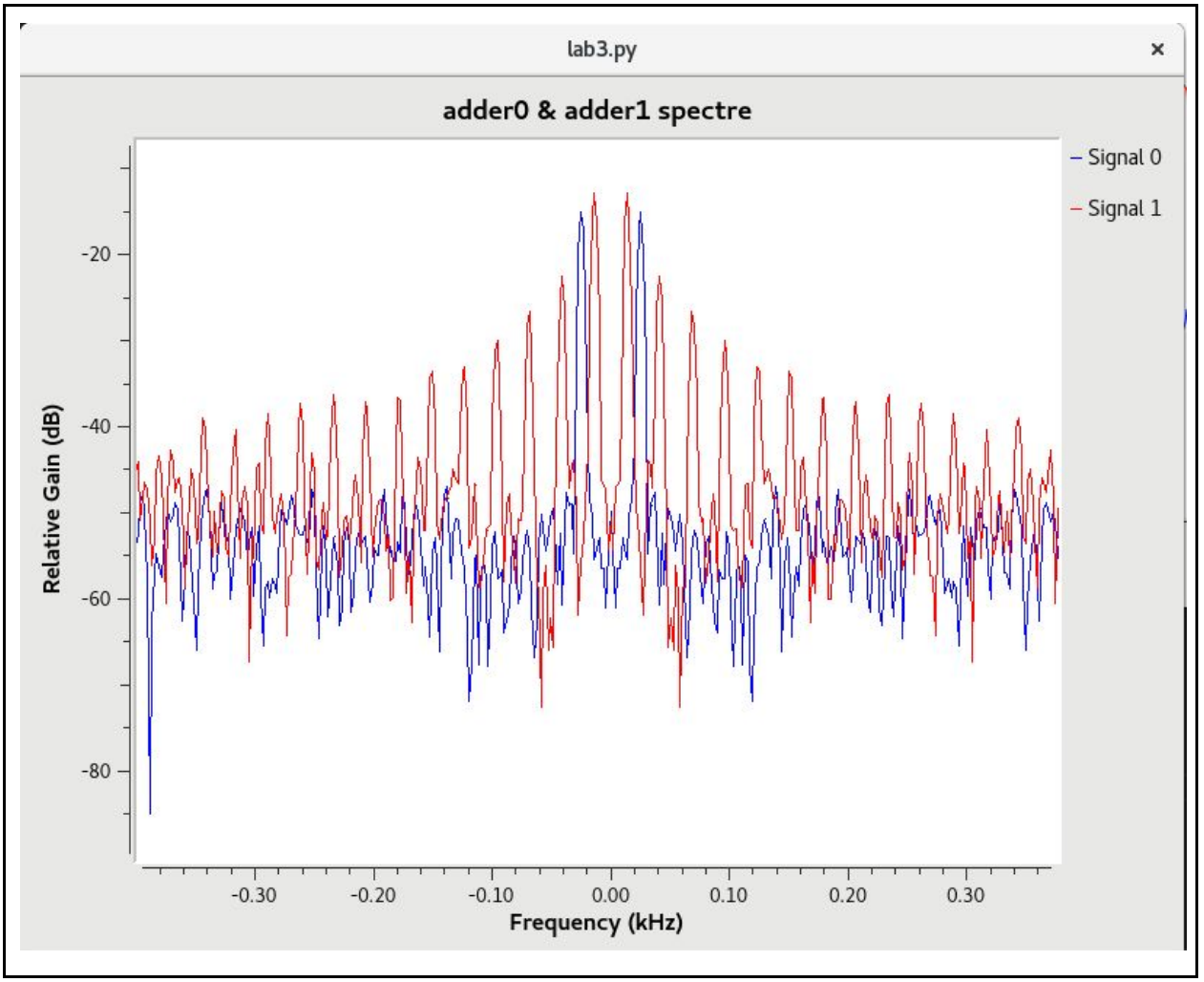
Además de esto, es posible insertarle a una señal ruido blanco gauseano aditivo, para lo cuál se implementará un sumador y una generador de ruido flotante, con las clases **“blocks.add_ff”** y **“analog.nouse_source_f”**, respectivamente.
También, a parte de graficar la señal en el tiempo como se hizo en la sección anterior, es posible graficar el espectro de la señal.
Uniendo lo anterior, a continuación se mestra el código que genera el flujograma descrito, generando una señal senoidal de 800 Hz y una señal cuadrada bipolar de 440 Hz, ambas de amplitud 1, a las cuales se le añade un AWGN de amplitud 0.1 (10% de la amplitud de las señales)

```
12  from gnuradio import gr, audio, analog, blocks
13  # Librerias para poder incluir graficas tipo QT
14  from gnuradio import qtgui
15  from PyQt4 import Qt # si no se acepta PyQt4 cambie PyQt4 por PyQt5
16  import sys, sip
17  ##################################################
18  # LA CLASE QUE DESCRIBE TODO EL FLUJOGRAMA
19  ##################################################
20  class my_top_block(gr.top_block):        # hereda de gr.top_block
21      def __init__(self):
22          gr.top_block.__init__(self)     # otra vez la herencia
23          # Parámetros:
24          ampl = 1
25          size = 500
26          sample_rate = 32000
27          plotName = "Signals from src0 & src1 with AWGN"
28          nconnections = 2
29          fftsize = 512
30          fc = 0
31          bw = 1000
32          plotFreqName = "adder0 & adder1 spectre"
33          # Bloques:
34          src0 = analog.sig_source_f(sample_rate, analog.GR_SIN_WAVE, 800, ampl)  #Generador 0
35          src1 = analog.sig_source_f(sample_rate, analog.GR_SQR_WAVE, 440, ampl*2, -ampl)  #Generador 1
36          noise0 = analog.noise_source_f(analog.GR_GAUSSIAN,(ampl*0.1))           #Generador de ruido 0
37          noise1 = analog.noise_source_f(analog.GR_GAUSSIAN,(ampl*0.1))           #Generador de ruido 1
38          adder0 = blocks.add_ff()                                                #Sumador 0
39          adder1 = blocks.add_ff()                                                #Sumador 1
40          dst = audio.sink(sample_rate, "")                                       #Reproductor de audio
41          self.snk = qtgui.time_sink_f(size,sample_rate,plotName,nconnections)    #Gracifa en tiempo
42          self.snk.set_line_label(0,"Signal 0")                                           #Asignar nombre
43          self.snk.set_line_label(1,"Signal 1")                                           #Asignar nombre
44          self.fsnk = qtgui.freq_sink_f(fftsize,5,fc,bw,plotFreqName,nconnections)#Gracifa en frecuencia
45          self.fsnk.set_line_label(0,"Signal 0")                                           #Asignar nombre
46          self.fsnk.set_line_label(1,"Signal 1")                                           #Asignar nombre
```

```
47          # Conecciones:
48          self.connect(src0, (adder0, 0)); self.connect(noise0, (adder0, 1))     #src0 y noise0 a las entradas de adder0.
49          self.connect(src1, (adder1, 0)); self.connect(noise1, (adder1, 1))     #src1 y noise1 a las entradas de adder1.
50          self.connect(adder0, (dst, 0))                                          #adder0 al reproductor.
51          self.connect(adder1, (dst, 1))                                          #adder1 al reproductor.
52          self.connect(adder0, (self.snk, 0))                                     #adder0 al graficador de tiempo.
53          self.connect(adder1, (self.snk, 1))                                     #adder1 al graficador de tiempo.
54          self.connect(adder0, (self.fsnk, 0))                                    #adder0 al graficador de frecuencia.
55          self.connect(adder1, (self.fsnk, 1))                                    #adder1 al graficador de frecuencia.
56          # Para mostrar gráfica:
57          pyWin = sip.wrapinstance(self.snk.pyqwidget(), Qt.QWidget)
58          pyWin.show()
59          pyWin2 = sip.wrapinstance(self.fsnk.pyqwidget(), Qt.QWidget)
60          pyWin2.show()
61  ##################################################
62  # EL CODIGO PARA LLAMAR EL FLUJOGRAMA "my_top_block"
63  ##################################################
64  def main():
65      qapp = Qt.QApplication(sys.argv)
66      myTop = my_top_block()
67      myTop.start()
68      qapp.exec_()
69      myTop.stop()
70  if __name__ == "__main__":
71      main()
```

Que genera como resultado las siguientes gráficas:




lab3.py
adder0 & adder1 spectre
Signal 0
Signal 1
Relative Gain (dB)
-20
-40
-60
-80
-0.30
-0.20
-0.10
0.00
0.10
0.20
0.30
Frequency (kHz)