Laboratorio 1: Programación en Radio Definida por Software (GNU Radio)

Carlos Stiven Roa Martinez
Universidad Industrial de Santander
Bucaramanga, Colombia
Email: carlos2201948@correo.uis.edu.co

Dana Cotes Cala
Universidad Industrial de Santander
Bucaramanga, Colombia
Email: dana2212256@correo.uis.edu.co

***Abstract*—In this lab report, the fundamental learning process for using an SDR began, specifically with GNU Radio and using GitHub.**

Various activities were carried out, one of which consisted of developing custom Python blocks with different functionalities, such as accumulation, differentiation, and evaluation of signal statistics. In addition, a practical application for these functionalities was sought to synthesize and demonstrate the knowledge acquired in this lab.

Repositorio: https://github.com/DamaGit/com2_A1_G-9

I. Introducción

El avance de las tecnologías en comunicaciones ha generado la necesidad de entornos de desarrollo flexibles que no solo permitan el uso de bloques predefinidos, sino también la creación de algoritmos propios. En este contexto, GNU Radio se presenta como una herramienta de radio definida por software (SDR) que facilita la implementación de sistemas de procesamiento de señales en tiempo real.

El presente laboratorio tiene como finalidad el desarrollo en competencias en la programación de funciones y bloques en GNU Radio, comprendiendo los aspectos fundamentales de los sistemas de tiempo real y de la SDR. De esta manera, se busca no solo utilizar los bloques ya existentes, sino también extender sus capacidades mediante la implementación de algoritmos como acumuladores, diferenciadores y funciones estadísticas, los cuales son útiles para el análisis y tratamiento de señales.

II. Objetivos

- Identificar los aspectos fundamentales en los sistemas de tiempo real y la radio definida por software.
- Generar funciones a partir de los bloques de implementación de código y evaluar los resultados con otros bloques.
- Utilizar los bloques implementados para producir una aplicación específica para señales reales.

III. Bloque Sumador

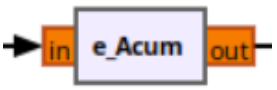

Fig. 1: Bloque Sumador

Calcula la suma acumulativa de las muestras de entrada en tiempo real. Es el equivalente discreto de una integral en tiempo continuo.

A. Configuración del bloque

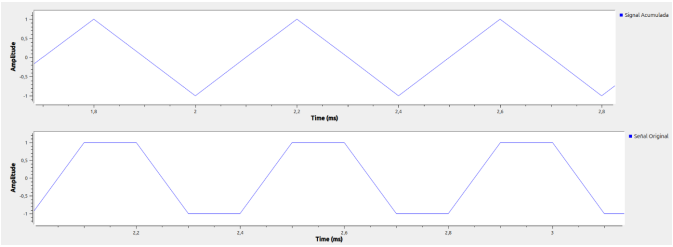

Fig. 2: Sumador / Integrador

- Un bloque programable que permite la entrada de un vector unidimensional y calcula la suma de sus datos a medida que se recorre la entrada, para este caso un vector con $\mu = 0$, con comportamiento de caja.
- Se obtuvo que para una entrada rectangular que varía entre -1 y 1, su integral es una función triangular en forma de diente de sierra, lo cual concuerda con la teoría.
- El bloque se puede desestabilizar con vectores que tengan $\mu \neq 0$, o también se puede obtener una salida acumulada desfasada, producto de un vector que tenga $\mu = 0$ pero cuyo primer valor sea considerado como una ganancia *DC* en el recorrido del mismo.

B. Código del bloque

En el listing 1 de la sección anexos se muestra cómo, para una señal de entrada, la salida del sistema es la suma acumulada de las componentes de entrada.

C. Aplicaciones

El bloque implementa un integrador digital universal y puede usarse en varias etapas de un sistema de comunicaciones:

- Demodulación: recuperación de información en esquemas donde la integración es necesaria.
- Generación de fase: útil en osciladores digitales y moduladores de fase/frecuencia.
- Filtrado acumulativo: reducción de ruido de alta frecuencia mediante integración.
- Detección de energía: evaluación de la energía acumulada para detección de paquetes o potencia recibida.
- Sincronización de lazo: aplicación en sistemas PLL y control de fase en receptores digitales.

Además, este bloque es útil en el análisis offline de señales para calcular la energía acumulada en el tiempo.

IV. Bloque Diferencial

Fig. 3: Bloque diferencial

Un bloque diferencial calcula la derivada discreta de una señal, es decir, mide el cambio entre una muestra y la anterior. Matemáticamente se expresa como:

$$y[n] = \frac{x[n] - x[n-1]}{T_s}$$

donde:

- $x[n]$ es la muestra actual de la señal.
- $x[n-1]$ es la muestra anterior.
- T_s es el período de muestreo.
- $y[n]$ es la salida diferencial.

Si la señal es constante ($x[n] = x[n-1]$), la salida es cero. Si hay un cambio, la salida refleja la magnitud y dirección de dicho cambio.

A. Configuración del bloque

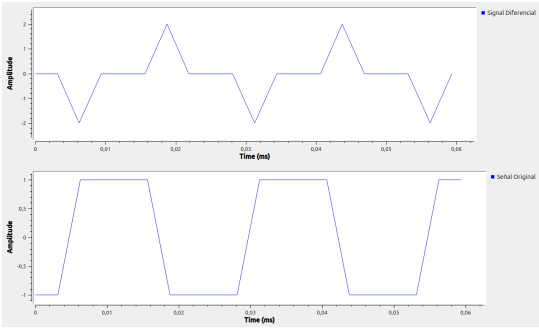

Fig. 4: Diferenciador / Derivador

El bloque diferenciador implementado calcula la diferencia entre cada muestra y la anterior, de acuerdo con la ecuación:

$$y[n] = x[n] - x[n-1]$$

Para la señal de entrada utilizada en la prueba:

$$x = [1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2]$$

la salida del bloque es:

$$y = [0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]$$

Es decir, mientras la señal de entrada permanece constante, la diferencia es cero; cuando ocurre una transición de 1 a 2, se genera un pico en la salida, y luego el valor vuelve a cero. Este comportamiento permite detectar cambios abruptos en la señal, siendo útil en aplicaciones de comunicaciones como detección de bordes, sincronización y demodulación.

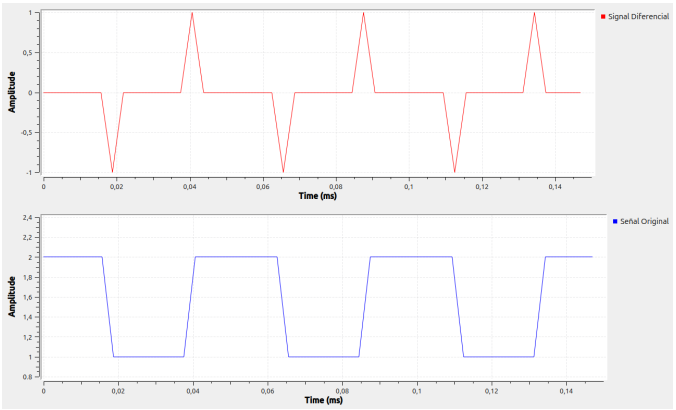

Fig. 5: Entrada [1,1,1,1,2,2,2,2,2]

B. Código del bloque

En el listing 2 de la sección anexos se muestra cómo, para una señal de entrada, la salida del sistema es la magnitud de la diferencia de los valores de entrada, mostrando una amplitud de 1 y sentido positivo, indicando un pico de incremento que puede ser detectado como una señal de información.

C. Aplicaciones

El bloque implementa un integrador digital universal y puede usarse en varias etapas de un sistema de comunicaciones:

- Demodulación: recuperación de información en esquemas donde la integración es necesaria.
- Generación de fase: útil en osciladores digitales y moduladores de fase/frecuencia.
- Filtrado acumulativo: reducción de ruido de alta frecuencia mediante integración.
- Detección de energía: evaluación de la energía acumulada para detección de paquetes o potencia recibida.
- Sincronización de lazo: aplicación en sistemas PLL y control de fase en receptores digitales.

Además, este bloque es útil en el análisis offline de señales para calcular la energía acumulada en el tiempo.

V. Bloque Estadístico

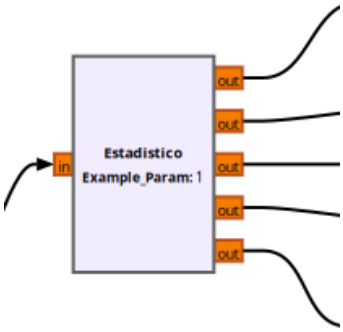

Fig. 6: Bloque estadístico

A. Configuración del bloque

El bloque Estadístico implementado en GNU Radio calcula en tiempo real diferentes parámetros estadísticos de una señal de entrada. El bloque recibe una señal $x[n]$ y produce cinco salidas:

- Promedio (media aritmética):

$$\bar{x} = \frac{1}{N_{\text{tot}}} \sum_{k=1}^{N_{\text{tot}}} x[k]$$

- Media cuadrática:

$$M_c = \frac{1}{N_{\text{tot}}} \sum_{k=1}^{N_{\text{tot}}} x^2[k]$$

- Valor RMS:

$$\text{RMS} = \sqrt{M_c}$$

- Potencia promedio:

$$P = (\text{RMS})^2 = M_c$$

- Desviaci ón estándar:

$$\sigma = \sqrt{\frac{1}{N_{\text{tot}}} \sum_{k=1}^{N_{\text{tot}}} \big(x[k] - \bar{x}\big)^2}$$

Para mantener continuidad entre ráfagas de datos, el bloque utiliza acumuladores que guardan las sumas anteriores y el total de muestras procesadas N_{tot}.

B. Código del bloque

En el listing 3 de la seción anexos, se puede apreciar el código de Python correspondiente al bloque estadístico.

C. Aplicaciones

El bloque Estadístico se integra fácilmente en flujos SDR para monitoreo y control en tiempo real. Usos típicos:

- Detección de canal (TVWS): decidir libre/ocupado con la potencia promedio P (media cuadrática). Conectar P a un *Comparator* y a un *QT GUI Number Sink*.
- Squelch adaptativo: abrir audio/decodificación sólo si $P > \mu + \alpha\sigma$. μ y σ provienen del propio bloque.
- Ajuste de ganancia / protección del ADC: usar RMS como objetivo para evitar *clipping*. Visualizar en *QT GUI* y ajustar ganancia del UHD/HackRF.
- Estimación rápida de SNR: medir P_{total} y P_{ruido} (sin portadora) y aproximar $\text{SNR}_{\text{dB}} \approx 10\log_{10}\big((P_{\text{total}} - P_{\text{ruido}})/P_{\text{ruido}}\big)$.
- Detección de ráfagas/paquetes: $E \approx N \cdot RMS^2$ en ventanas deslizantes para disparar el *Packet Header Parser* o abrir compuertas (*Gate*).
- Verificación de modulaciones: comprobar normalización tras el *Constellation Decoder*; un RMS desviado sugiere desbalance I/Q o escalado incorrecto.
- Diagnóstico de clipping (factor de cresta): combinar RMS con un *Peak Detector* externo para calcular $CF = x_{\text{pico}}/RMS$ y detectar compresión en PA/ADC.
- Evaluación de filtros y cadena RF: comparar P antes/después de un filtro para estimar atenuación en banda y rechazo fuera de banda.
- Perfilado de ruido/interferencias: registrar P y σ en un *File Sink* para analizar ocupación por horario y variabilidad del canal.
- Control por estados: usar σ baja como indicio de bloqueo del Costas Loop/PLL y conmutar entre rutas o parámetros del ecualizador.

VI. Sistema en conjunto

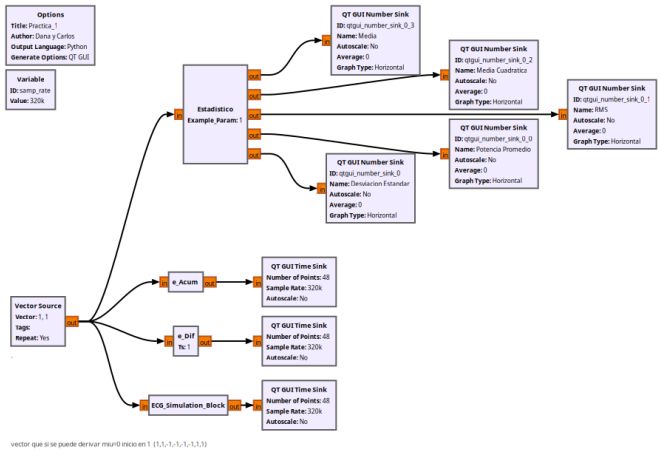

Fig. 7: Salida del sistema

A. Configuración del sistema

La Fig. 7 muestra el flujograma completo en GNU Radio. Los parámetros usados fueron:

- Variable general: `samp_rate` = 320 kS/s.
- Vector Source: `Vector` = $[+1, -1, +1, -1, \ldots]$.
- Bloque Estadístico: entrada `float32`; cinco salidas: media $(\bar{x})$, media cuadrática (M_c), RMS $(\sqrt{M_c})$, potencia promedio (P) y desviación estándar (σ).
- QT GUI Number Sinks (5): uno por cada salida del bloque, con nombres *Media*, *Media Cuadrática*, *RMS*, *Potencia Promedio* y *Desviación Estándar*.
- e_Acum (acumulador): entrada desde el *Vector Source*;

salida a un *QT GUI Time Sink*.

- e_Dif (diferenciador): entrada desde el *Vector Source*; salida a un *QT GUI Time Sink*.
- ECG_Simulation_Block: generador sintético para contraste visual; salida a un *QT GUI Time Sink*.
- QT GUI Time Sinks (3): para visualizar en el tiempo.

B. Aplicación: Señales ECG con Ruido Gaussiano

A partir del sistema mostrado en la Figura 8, es posible modelar una aplicación de la vida real como lo es un simulador de ECG; a este se le aplica un ruido gaussiano con media cero (bloque *Noise Source*) que asemeja la realidad de estas señales. A continuación se muestra el flujograma, con los respectivos resultados del sistema.

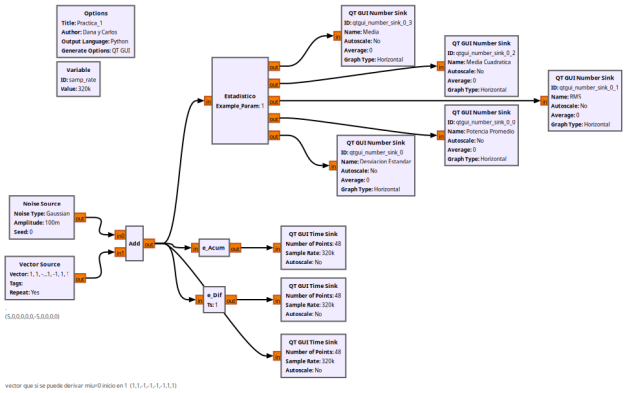

Fig. 8: Sistema de lectura ECG

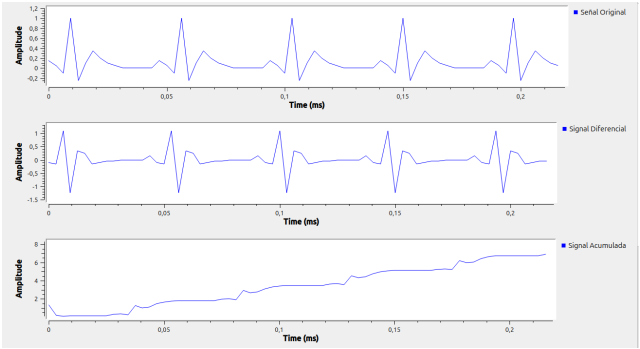

Fig. 9: Señales diferencial y acumulada.

La gráfica 9 muestra cómo, a partir de una señal de entrada, el bloque acumulador genera la integral discreta (señal acumulada) mientras que el bloque diferenciador obtiene la variación puntual (señal diferencial). Este contraste permite observar el comportamiento complementario de ambos bloques en el dominio temporal.

En esta figura se presentan parámetros clave de la señal procesada: promedio, media cuadrática, valor RMS, potencia promedio y desviación estándar. Estos datos permiten cuantificar la energía de la señal, medir su dispersión y evaluar la potencia efectiva, lo cual resulta esencial en aplicaciones de comunicaciones para la estimación de SNR, ajuste de ganancia y caracterización de ruido en canales de transmisión.

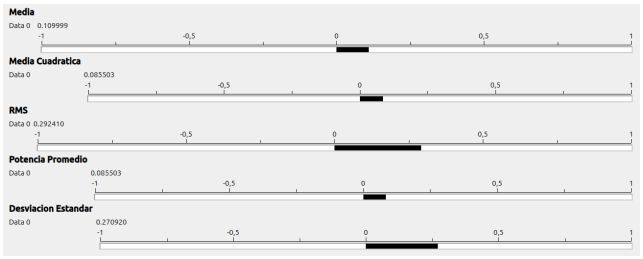

Fig. 10: Valores estadísticos.

La Figura 10 muestra los valores estadísticos obtenidos de la señal de entrada. En primer lugar, la **media** es de aproximadamente 0.11, lo cual indica la presencia de una pequeña componente DC en la señal; en un caso ideal sin sesgo, este valor debería tender a cero.

La **media cuadrática** y la **potencia promedio**, ambas con un valor cercano a 0.0855, cuantifican la energía media de la señal y reflejan su intensidad en el tiempo.

El valor **RMS** obtenido es 0.2924, representando la magnitud eficaz de la señal, mayor que la media, lo cual confirma la existencia de variaciones importantes alrededor del valor promedio.

Finalmente, la **desviación estándar** es de 0.2709, lo que señala una dispersión considerable respecto a la media, característica asociada a la presencia de ruido y variabilidad en la señal.

En conjunto, estos resultados indican que la señal presenta un leve sesgo DC, una potencia moderada y un nivel significativo de variabilidad, comportamiento esperado en señales aleatorias sometidas a ruido.

VII. Conclusiones

En este laboratorio se comprendieron y aplicaron herramientas clave de radio definida por software (SDR) mediante GNU Radio y bloques implementados en Python. Se desarrollaron y evaluaron tres bloques —acumulador, diferenciador y estadístico— con los cuales se analizó el comportamiento de señales en distintos escenarios. Como aplicación práctica, se replicó el complejo PQRST de un electrocardiograma (ECG) y se estudiaron sus parámetros estadísticos, evidenciando la utilidad de estos bloques en el procesamiento de señales. Los resultados validan el correcto funcionamiento de los sistemas implementados y refuerzan la pertinencia de la SDR en telecomunicaciones y en el análisis de señales. En conjunto, la experiencia fortaleció las competencias en diseño y prueba de herramientas de procesamiento digital de señales y abre la puerta a aplicaciones más avanzadas en ingeniería electrónica.

- El bloque estadístico permitió evaluar métricas fundamentales como promedio, RMS y potencia, herramientas clave en comunicaciones para caracterizar señales con ruido.
- Se comprobó el correcto funcionamiento de los bloques diseñados en *GNU Radio*, logrando implementar operaciones de acumulación, diferenciación y análisis estadístico.

- Los resultados experimentales coinciden con la teoría: la integración de una señal rectangular genera una forma triangular, mientras que la diferenciación refleja los cambios instantáneos de la señal.
- Se hace hincapié en la generación de diseños de filtros a partir de los resultados estadísticos, basados en cómo las perturbaciones alteran al sistema.

VIII. Anexos

El código y archivos asociados a este trabajo se encuentran disponibles en el siguiente repositorio de GitHub:

- https://github.com/DamaGit/com2_A1_G-9

Listing 1: Sistema acumulador

```
import numpy as np
from gnuradio import gr

class blk (gr.sync_block ):
    def __init__( self ) : # only default arguments here
        gr.sync_block.__init__(
            self ,
            name ="e_Acum ", # will show up in GRC
            in_sig =[np.float32],
            out_sig =[np.float32]
        )

    def work (self,input_items,output_items):
        x = input_items [0] # Senial de entrada .
        y0 = output_items [0] # Senial acumulada
        y0 [:] = np.cumsum(x)

        return len (y0)
```

Listing 2: Sistema diferenciador

```

import numpy as np
from gnuradio import gr

class blk(gr.sync_block):
    def __init__(self):
        gr.sync_block.__init__(
            self,
            name='e_Dif',
            in_sig=[np.float32],
            out_sig=[np.float32]
        )
        self.acum_anterior = 0

    def work(self, input_items, output_items):
        x = input_items[0]  # Se al de entrada
        y = output_items[0]  # Se al de salida
        N = len(x)

        # Diferenciaci n acumulada: y[n] = x[n] - acum_anterior
        for i in range(N):
            y[i] = x[i] - self.acum_anterior  # Restamos el valor acumulado anterior
            self.acum_anterior = x[i]  # Actualizamos el acumulado con el valor actual

        return len(y)
```

Listing 3: Sistema estadístico

```
import numpy as np
from gnuradio import gr

class blk(gr.sync_block):
    def __init__(self, example_param=1.0):
        gr.sync_block.__init__(
            self,
            name='Promedios_de_tiempo',
            in_sig=[np.float32],
            out_sig=[np.float32,np.float32,np.float32,np.float32,np.float32]
        )
        self.acum_anterior = 0
        self.Ntotales = 0
        self.acum_anterior1 = 0
        self.acum_anterior2 = 0

    def work(self, input_items, output_items):
        x=input_items[0]    #se al de entrada
        y0=output_items[0]  #promedio de la se al de entrada
        y1=output_items[1]  #media de la se al
        y2=output_items[2]  #RMS de la se al
        y3=output_items[3]  #potencia promedio de la se al
        y4=output_items[4]  #desviaci n estandar de la se al

        #promedio
        N = len(x)
        self.Ntotales = self.Ntotales + N
        acumulado = self.acum_anterior + np.cumsum(x)
        self.acum_anterior = acumulado[N-1]
        y0[:] = acumulado/self.Ntotales

        #media cuadratica
        x2=np. multiply (x,x)
        acumulado1 = self.acum_anterior1 + np.cumsum(x2)
        self.acum_anterior1 = acumulado[N-1]
        y1[:] = acumulado1/self.Ntotales
        #RMS
        y2[:]=sqrt(y1)
        #potencia promedio
        y3[:]=np.multiply(y2,y2)
```

```
        #desviaci n estandar
        x3 = np.multiply(x-y0 ,x-y0)
        acumulado2 = self.acum_anterior2 +
            np.cumsum(x3)
        self.acum_anterior2 = acumulado2[N
            -1]
        y4[:] = np.sqrt(acumulado2/self.
            Ntotales)

        return len(x)
```

REFERENCES

[1] Comunicaciones Plus, *Comunicaciones Plus Volumen II*, Comunicaciones Plus, 2022.

[2] GitHub, "Repositorio del proyecto COM2_A1_G-9," Available: https://github.com/DamaGit/com2_A1_G-9.

[3] OpenAI, "ChatGPT: Language Model for Conversational AI," Available: https://openai.com/chatgpt.

[4] YouTube, "El VALOR MEDIO de las Señales," Available: https://youtu.be/vstBIp2cxcw?si=JauegdDaHJ_NOEQK, 2020.