# Práctica 1: GIT, GITHUB Y GNURADIO

FABIO ANDRES CORZO ARGUELLO - 2180383
DIEGO ANDRES ARDILA ARIZA - 2185594
CARLOS HUMBERTO DIAZ SALAZAR -2182353

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones
Universidad Industrial de Santander

6 de septiembre de 2023

**Resumen**

El objetivo principal de esta práctica de laboratorio es brindar a los estudiantes la oportunidad de adquirir habilidades fundamentales en el manejo de dos herramientas esenciales: GitHub y el software GNU Radio. Para alcanzar este propósito, se estableció un repositorio en GitHub en el cual cada estudiante se involucró en la exploración y aplicación de diversos comandos, permitiéndoles comprender en profundidad el funcionamiento de GitHub. Posteriormente, se procedió a la creación de bloques en GNU Radio, tales como el multiplicador, el acumulador y el diferenciador. Este enfoque facilitó a los participantes adquirir habilidades sólidas tanto en el uso de GitHub como en la programación con GNU Radio."

## 1. Introducción

En el contexto de las comunicaciones y el procesamiento de señales, la capacidad de desarrollar bloques de procesamiento personalizados es importante para adaptar y mejorar el rendimiento de los sistemas. La herramienta GNU Radio, con su versatilidad y flexibilidad, se ha convertido en una plataforma poderosa para diseñar, simular y ejecutar flujos de señales de manera eficaz. Sin embargo, la creación de bloques personalizados en Python, es una tarea fundamental para aprovechar al máximo esta herramienta.

Este informe presenta un proyecto en el que utilizamos GitHub, es una forja (plataforma de desarrollo colaborativo) para alojar proyectos utilizando el sistema de control de versiones Git [1]. Se utiliza principalmente para la creación de código fuente de programas de ordenador. para gestionar y versionar nuestro código, y GNU Radio, una herramienta de desarrollo libre y abierta que provee bloques de procesamiento de señal para implementar sistemas de radio definida por software [2], para crear bloques personalizados en Python. Los bloques en cuestión son el diferenciador, el acumulador y el multiplicador, todos diseñados para operar con vectores de datos de entrada.

A través de este informe, se va explorando cómo GitHub facilita la colaboración y el control de versiones en el desarrollo de software, garantizando la integridad y el control de versiones de nuestros bloques personalizados. Además, se analizará la implementación y el funcionamiento de los bloques diferenciador, acumulador y multiplicador, destacando su importancia en aplicaciones de procesamiento de señales y comunicaciones.

## 2. Procedimiento

- **Multiplicador:** Este bloque tiene la función de multiplicar una señal de entrada, que es un vector de datos, por una constante que puede configurarse mediante un parámetro. A continuación, desglosamos las características esenciales de este bloque:

  **Nombre del Bloque y Parámetros:**

  El bloque se denomina Embedded Python Block y este nombre es fundamental, ya que será utilizado para identificar y configurar el bloque dentro del entorno de GNU Radio Companion (GRC). Además, cuenta con un parámetro llamado `example_param`, que representa la constante que se utilizará para multiplicar la señal de entrada. Este

parámetro puede configurarse desde GRC y, de esta manera, controlar el comportamiento del bloque de manera dinámica.

### Inicialización del Bloque:

En el método `__init__`, se lleva a cabo la inicialización del bloque y se configuran sus parámetros. Aquí se especifica que el bloque tiene una señal de entrada de tipo `np.float32` y una señal de salida de tipo `np.float32`. Estos tipos de datos son comunes en el procesamiento de señales. Además, se asigna el nombre Embedded Python Block al bloque, lo que facilita su identificación en GRC. Asimismo, se asigna el valor del parámetro `example_param`, utilizando el valor proporcionado al crear una instancia del bloque.

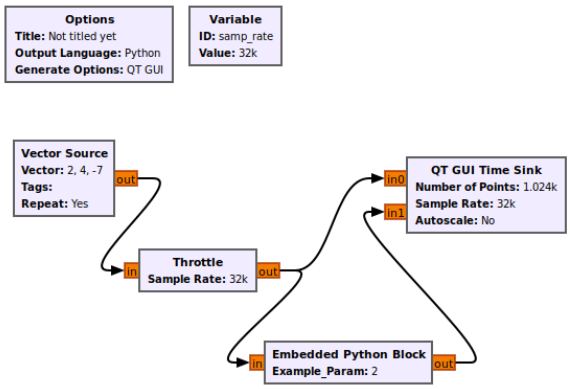


**Fig 1.** Bloques Multiplicador

### Función de Trabajo (`work`):

El método `work` es el núcleo del bloque, donde se realiza la operación principal de multiplicación por la constante. La señal de entrada `input_items` se multiplica elemento por elemento por el valor de `example_param`. El resultado se almacena en la señal de salida `output_items`. Posteriormente, la función devuelve la longitud de la señal de salida, que generalmente coincide con la longitud de la señal de entrada.

```python
import numpy as np
from gnuradio import gr


class blk(gr.sync_block):  # other base classes are basic_block, decim_block, interp_block
    """Embedded Python Block example - a simple multiply const"""

    def __init__(self, example_param=1.0):  # only default arguments here
        """arguments to this function show up as parameters in GRC"""
        gr.sync_block.__init__(
            self,
            name='Embedded Python Block',   # will show up in GRC
            in_sig=[np.float32],
            out_sig=[np.float32]
        )
        # if an attribute with the same name as a parameter is found,
        # a callback is registered (properties work, too).
        self.example_param = example_param

    def work(self, input_items, output_items):
        """example: multiply with constant"""
        output_items[0][:] = input_items[0] * self.example_param
        return len(output_items[0])
```

**Fig 2.** Código bloque Multiplicador

### Uso en un Flujo de Señal:

Para utilizar este bloque en un flujo de señal de GNU Radio, primero se debe instanciar el bloque y configurar sus parámetros, incluyendo `example_param`. Luego, se puede conectar a otros bloques, como un bloque de fuente de datos (vector) y un bloque Throttle (para controlar la velocidad de muestreo). Finalmente, la señal procesada se puede visualizar utilizando un bloque QT GUI Time Sink.

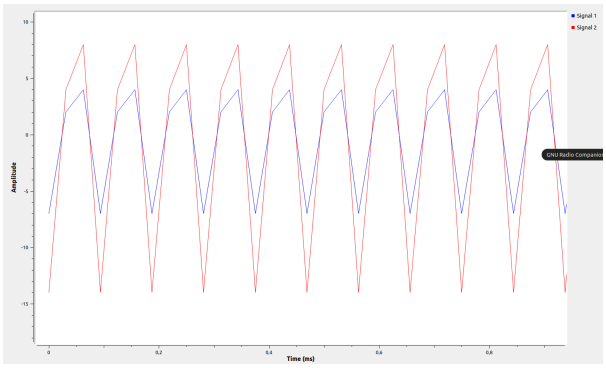

**Fig 3.** Time Sink Multiplicador

- Acumulador: El bloque Acumulador en GNU es muy importante ya que este nos permite acumular o sumar las muestras de una señal de entrada para diversas funciones ya sea para calcular un promedio de la señal de entrada o para integrar una señal continua. Para implementar este bloque usamos el siguiente código:

```python
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 ='Acumulador ', # will show up in GRC
            in_sig =[ np. float32 ],
            out_sig =[ np. float32 ]
        )
    def work(self, input_items, output_items):
        """example: multiply with constant"""
        output_items[0][:] = np.cumsum(input_items[0])
        return len(output_items[0])
```

**Fig 4.** Código bloque Acumulador

Donde podemos observar que inicializamos los datos de entrada como flotantes y los de salida igual que los datos de salida van a ser el primer dato de entrada acumulado del siguiente dato de entrada haciendo que esto siempre se sume. una vez creado el bloque procedemos hacer el montaje para poder observar la acumulación procedemos a crear el siguiente programa para la obtención de datos y la acumulación de los mismos:

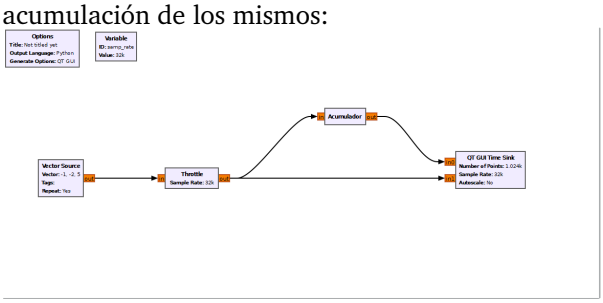


**Fig 5.**Implementación de acumulador

Donde podemos observar que al correr los bloques observamos dos señales, la señal 1 que es la acumulada y la señal 2 que son los datos sin acumulada, aquí podemos observar que la señal 1 tiende hasta la suma de todas las mismas con un pequeño reinicio, pero por otro lado la señal dos observamos que es una señal que oscila entre -1 hasta 5 y así sucesivamente.

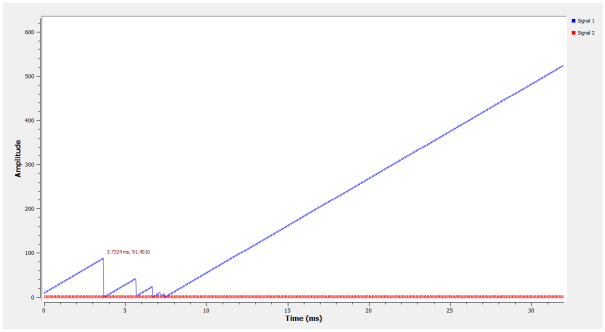


**Fig 6.**Resultados acumulador de acumulador

- Diferenciador: El bloque diferenciador en GNU Radio desempeña un papel fundamental en el análisis de señales al calcular la derivada discreta de una secuencia de datos de entrada. Este proceso implica restar cada elemento de la secuencia actual de datos del elemento anterior, generando así una nueva secuencia de datos de salida que representa las tasas de cambio entre puntos adyacentes [3]. En términos matemáticos, si consideramos una secuencia de entrada como $x[n]$, el bloque diferenciador genera una secuencia de salida $y[n]$ mediante la fórmula $y[n] = x[n] - x[n-1]$.
Utilizando el bloque Python provisto por GNU Radio, se implementó el siguiente código para la creación del bloque diferenciador:

```python
import numpy as np
from gnuradio import gr

class blk(gr.sync_block):

    def __init__(self):  # Solo argumentos predeterminados aquí
        gr.sync_block.__init__(
            self,
            name='Diferenciador',  # Aparecerá en GRC
            in_sig=[np.float32],
            out_sig=[np.float32],
        )
        self.acum_anterior = 0

    def work(self, input_items, output_items):
        x = input_items[0]  # Señal de entrada.
        y0 = output_items[0]  # Señal acumulada diferencial
        N = len(x)
        diff = np.cumsum(x) - self.acum_anterior
        self.acum_anterior = diff[N - 1]
        y0[:] = diff
        return len(y0)
```

**Fig 7.** Código bloque diferenciador

Como se observó en el código anterior, una señal de entrada en formato de valores de punto flotante fue procesada mediante un bloque de diferenciación acumulativa implementado en GNU Radio utilizando Python. En el inicio de la operación, el valor acumulado previo se inicializó a cero. Dentro del método "work", se efectuó el cálculo de la diferencia acumulativa mediante la aplicación de

la suma acumulada a la señal de entrada, seguida de la sustracción del valor acumulado anterior. Cabe destacar que esta operación se ejecutó eficientemente empleando la función "np.cumsum"de la biblioteca NumPy. El resultado de este procedimiento se asignó directamente a la señal de salida, a la vez que se actualizó el valor acumulado previo, listo para su utilización en el ciclo de procesamiento subsiguiente.

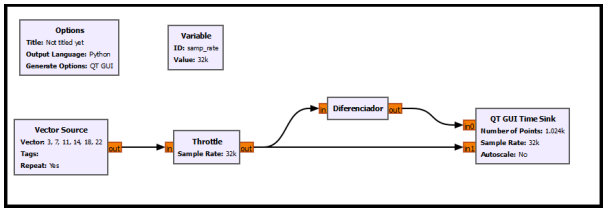


**Fig 8.** Diagrama de bloques diferenciador

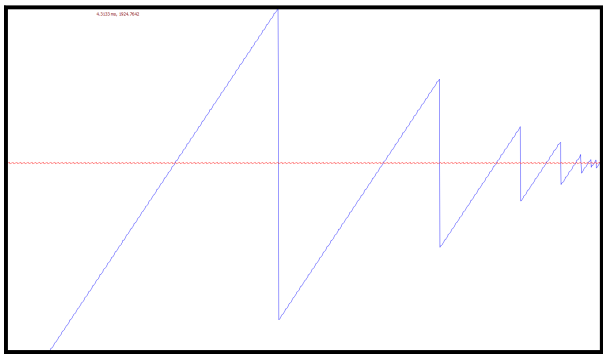

**Fig 9.** Resultado diagrama de bloques diferenciador

## 3. Conclusiones

- La implementación de un multiplicador personalizado proporciona una comprensión práctica de cómo diseñar, configurar y utilizar bloques en GNU Radio. La flexibilidad para definir parámetros y operaciones específicas permite la creación de bloques adaptados a aplicaciones concretas.
- El bloque acumulador en GNU Radio es una herramienta esencial para sumar y procesar muestras de señales a lo largo del tiempo, lo que resulta útil en una variedad de aplicaciones de procesamiento de señales y comunicaciones. Su configuración adaptable lo hace versátil para tareas como mejorar la detección de señales débiles y calcular estadísticas en tiempo real.
- El bloque diferenciador desempeña un papel importante gracias a la capacidad para calcular la derivada discreta de secuencias de datos resulta fundamental para detectar cambios rápidos en las señales, lo que puede ser crucial en la demodulación, la corrección de errores y la detección de eventos importantes en transmisiones de comunicaciones

## Referencias

[1] “Github.” [Online]. Available: es.wikipedia.org/wiki/GitHub

[2] “Gnuradio.” [Online]. Available: es.wikipedia.org/wiki/GNU_Radio

[3] A. V. Oppenheim, “Tratamiento de señales en tiempo discreto,” 2011.