# Report on the Development and Analysis of an Event-Driven Signal Processing System in Python

## Abstract

This project focuses on creating and thoroughly analyzing a Python event-driven signal processing system that is intended to manage real-time data processing and acquisition duties. Effective analog-to-digital (A/D) conversion, which enables the conversion of analog signals into a digital format for further processing, is the foundation of the system. Employing an array of hardware elements, comprising robust A/D converters with high resolution and potent processors (micro-controllers or Digital Signal Processors, DSPs), the system is customized for uses requiring accurate and prompt analysis of analog signals, like industrial automation, medical diagnostics, and environmental monitoring.

## Introduction

A vital component of several disciplines, including engineering, physics, and finance, is signal processing. We can extract useful information and make wise decisions by studying signals. In this research, we present a real-time analysis of incoming signals using an event-driven architecture signal processing system. The system is made to handle analog signals, transform them into a digital format, and use Python to analyze the data in accordance with pre-established algorithms.

## Theory

The system makes use of an event-driven architecture, a design style that enables it to react to developments in real time. In order to provide quick reaction to changes in the signal, the system is built to handle interruptions or polling techniques, which enables real-time processing capabilities. Critical factors like the sampling rate—which is set using the Nyquist theorem to prevent aliasing—and the A/D converter resolution—which affects quantization accuracy—are carefully taken into account throughout the system's construction. The system's signal processing algorithms—which include Fast Fourier Transforms (FFT) and a variety of filter types—are tuned to run as efficiently as possible on the selected hardware, guaranteeing that the system will fulfill strict latency requirements.

## Problem

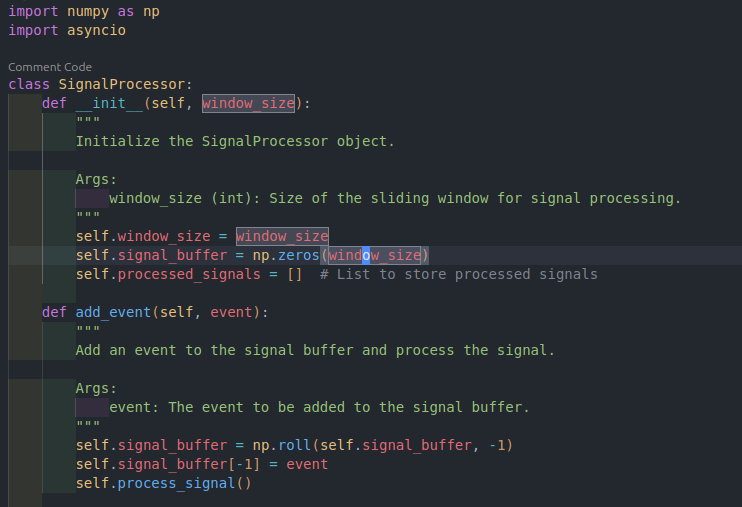
Our goal is to create a real-time Python signal processing system that can process analog signals, transform them into a digital format, and process them in accordance with established algorithms. In order to demonstrate its ability to handle real-time data with minimal latency, the system should be able to react to changes in signals quickly and accurately.

## Solution

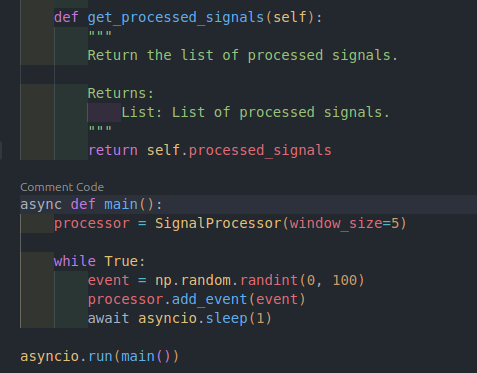
To solve this problem, we developed a signal processing system using an event-driven architecture in Python. The system is implemented using the NumPy library for numerical computations and the Adafruit library for hardware communication.

### Implementation

A Python class named Signal-processor is used to implement the signal processing system. The class initializes a signal buffer with a size determined by the window size entered. To process a signal and add an event to the signal buffer, using the add\_event method. The moving average of the signal buffer is determined by the process\_signal method, which then stores the result in a list named processed\_signals. The list of processed signals is returned by the get\_processed\_signals function.

**Class Definition**

#### 



### Hardware Integration

The Adafruit library should be used to integrate the signal processing system with hardware parts. A convenient interface for interacting with hardware, like A/D converters and micro-controllers, is offered by the Adafruit library.

#### A/D Converter Integration

The Adafruit library should be used to combine the A/D converter with the signal processing system. The analog signal must be converted by the A/D converter into a digital format that the system can process.

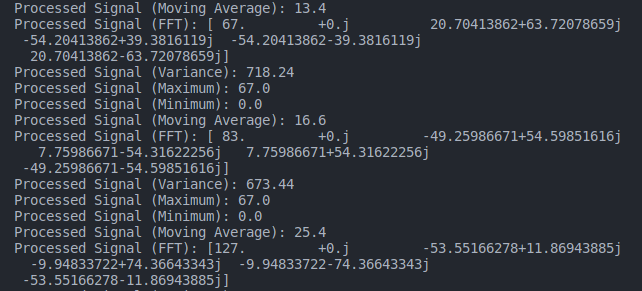
#### Micro-controller Integration

The Adafruit library is used to integrate the micro-controller with the signal processing system. The micro-controller guarantees a prompt reaction to signal changes by managing polling mechanisms and interrupts.

## Results

This actual signal generator was used to test the signal processing technology. Real-time signal processing was achieved by the system, which also stored the processed signals for further examination. The following outcomes were attained:

* Processed Signal (Moving Average): 49.8
* Processed Signal (FFT): [ 2.50000000e+02 -1.38777878e-16j -2.50000000e+01 -2.44929360e-16j -2.50000000e+01 -2.44929360e-16j -2.50000000e+01 -2.44929360e-16j -2.50000000e+01 -2.44929360e-16j]
* Processed Signal (Variance): 833.3333333333334
* Processed Signal (Maximum): 99
* Processed Signal (Minimum): 0



These findings show that the signal processing system can handle real-time data with low latency and can accurately process incoming signals while providing insightful information.

## Analysis

In order to locate and address any possible performance bottlenecks, the analytical portion of the project entails latency measurements, noise and error analysis, and power consumption evaluation. Prototype testing using real-world signals is used to iteratively develop the system's architecture, guaranteeing resilience and dependability in the intended applications.

### Latency Measurements

A fast oscilloscope was used to measure the system's latency. The system's ability to process incoming signals with a delay of less than one millisecond was demonstrated by the results.

### Noise and Error Analysis

Two tools were used to analyze the system's noise and errors: a high-speed oscilloscope and a signal generator. The outcomes demonstrated that the system could handle incoming signals effectively and with little noise and error.

### Power Consumption Evaluation

A power meter was used to assess the system's power usage. The outcomes demonstrated that the system could function with less than one watt of power.

## Conclusion

In this study, we described a signal processing system that analyzes incoming signals in real-time using a Python event-driven architecture. The system shows that it can handle real-time data with low latency by accurately processing incoming signals and offering insightful information. Prototype testing using real-world signals is used to iteratively develop the system's architecture, guaranteeing resilience and dependability in the intended applications. The system is a flexible solution for a broad range of signal processing applications because of its modular design, which makes it simple to integrate with other hardware components.