openDAQ assignment
IIR Filter Function Block

Due date:
February 27, 2026

---

1 Introduction

The assignment entails the creation of a low-pass Butterworth IIR filter function block that accepts a synchronous signal as its input and outputs the filtered signal. The function block is to be implemented as a stand-alone module (shared library), using the openDAQ library as an external source, following the example provided in the Available resources section..

The assignment is open-ended, meaning that any libraries and tools can be used to complete the tasks. To provide structure to the assignment, a list of high-level steps is available in the Tasks section. It is up to your discretion to decide up to which task you wish to complete the assignment.

2 Available resources

To aid in your task progression, the following resources, available at the Developement documentation/downloads webpage, should be used:

- **Documentation:** The documentation articles is featured in the "User guide" tab of the web page contains how-to guides, tutorials and information on the SDK architecture. The API documentation is available in the "API reference" tab.
- **Examples:** C++ and Python examples for different application use cases of the SDK are available in SDK web page folder, as well as in the openDAQ repository.
- **Simulator:** openDAQ simulator virtual image that runs a simulated device server on boot.

The openDAQ source code is available at `https://github.com/openDAQ/openDAQ`. Several module implementations are available in the "openDAQ/modules" folder that can be used as a reference for your tasks.

You should implement your function block as a module that uses openDAQ as an external library. An example of such a module is available in the SimpleFBModule repository, the repository contains a modified version of the Scaling function block available in the main openDAQ repository.

2.1 Asking questions

If you would like any support, clarifications, or if you get stuck at some point in the assignment, the openDAQ SDK team is available to answer questions via email. You can direct any questions regarding the assignment to "dusan.kalanj@opendaq.com". For longer questions, a remote meeting can also be arranged.

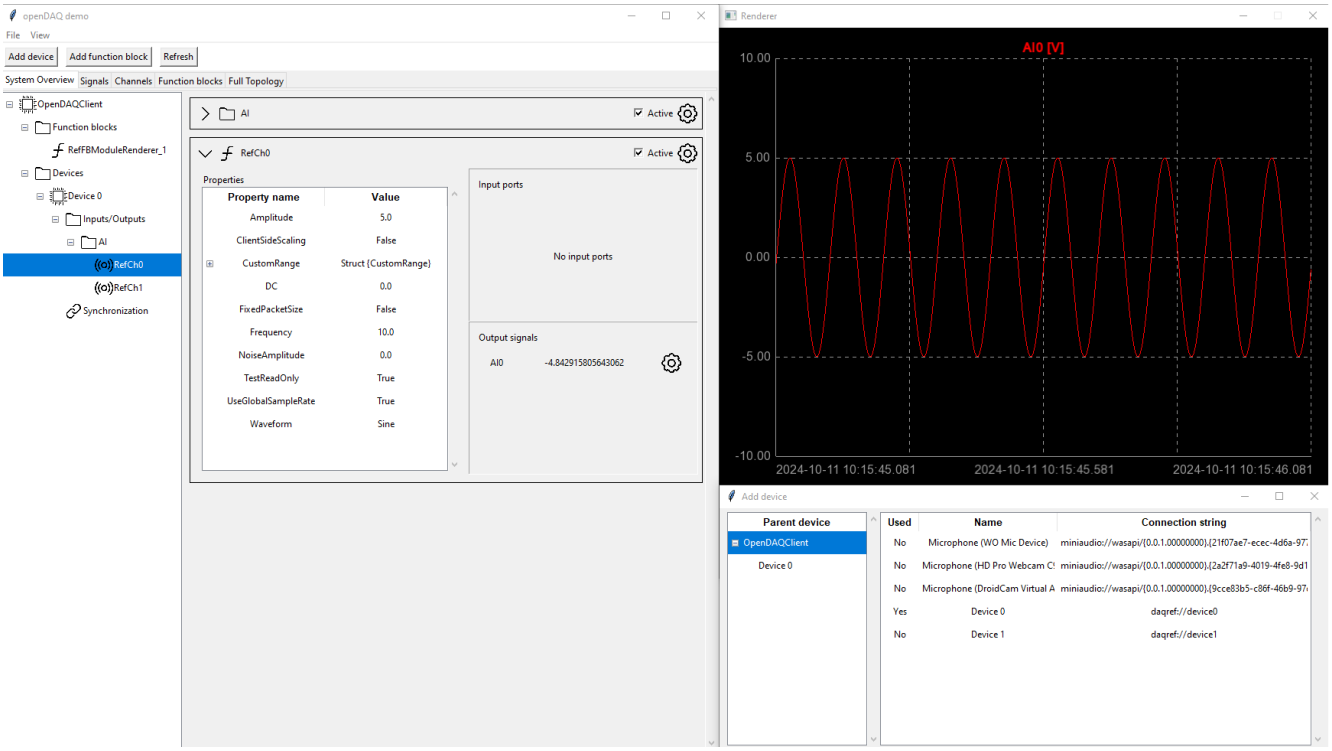

Figure 1: Screenshot of the Python GUI application with an added reference device and visualised output signal

2.2 Simulating data

To simulate data locally, a simulated "Reference device" can be added that outputs two sine-wave signals. To add noise to its sine signals, each "Channel" of the device has a property called "NoiseAmplitude" that can be adjusted to add noise to the outputs.

To visualise the data, the Python GUI app can be used, or, as illustrated in the openDAQ documentation Quick start guide, you can add the "Renderer" function block via:
`instance.addFunctionBlock("RefFBModuleRenderer");`

2.3 Using the Python GUI application

The Python GUI application allows for quick connection to an openDAQ device and allows for the visualisation of signals. The application is available at the openDAQ repository path "examples/applications/python/GUI Application/gui_demo.py". To run the application, you should install one of the Python wheels. Using the "Add device" button, you will be able to add the openDAQ Simulator device, whereas the "Add function block" button will let you add any reference function block, or later, your own.

To load your module in the Python application, the application should be run with an additional command line argument "–module_path" that points to your build directory. Make sure to build your project with the "Release" target when doing so.

3 Tasks

Before starting, you are recommended to follow the openDAQ Application Tutorial to get to know the SDK and its intended uses.

The example scaler in the SimpleFBModule repository should be used as a reference when attempting the below tasks.

The tasks are as follows:

1. Clone and build the the SimpleFBModule.
2. Extend the module to allow for adding a new function block via `addFunctionBlock()` with the ID ‘ExampleIIRFilter‘. The list of function block types returned via ‘getAvailableFunctionBlockTypes()‘ should contain a type with said ID.
3. Add an input port that accepts a scalar signal with a domain signal with an implicit, linear data rule.
4. Attach a Stream reader to the input port, and configure its "Data available" callback by passing a lambda that invokes your calculation function via ‘reader.setOnDataAvailable()‘
5. Add logic to process the event packets of the signal connected to the input port, validating that the input data signals contain scalar data, and that their domain signals have an implicit, linear data rule.
6. Add logic to process the data packets of the signal connected to the input port, filtering them with a low-pass IIR Butterworth filter. Output the filtered data as packets into the output signal. You can use the "Renderer" function block to visualise the output data and the "Reference device" or simulator to simulate noisy sine wave signals.
7. Add a property to the function block that controls the frequency cutoff of the low-pass filter. The property should be named "CutoffFrequency" and should accept an integer-type value.
8. Enable test for the SimpleFBModule repository and add additional tests to validate the behaviour of your filter.

4 How to submit

To submit the assignment solution, upload the folder containing your function block implementation to a cloud storage server provider of your choosing (Google Drive, OneDrive), and share it with "dusan.kalanj@opendaq.com". The example should contain an executable that sets up your function block with an input signal of the reference device connected to its input port.