System Documentation: Real-Time Signal Processing Client and Server

1. System Overview

Purpose: This system provides a robust framework for prototyping and validating digital signal processing (DSP) algorithms. It consists of a server application that transmits raw signal data and a client application that receives, processes, analyzes, and visualizes this data in real-time.

The client's core functionality is built around Arm's **CMSIS-DSP library**, allowing for high-performance signal processing logic to be developed and tested in a flexible Python environment before being deployed to embedded hardware. The primary application demonstrated is the real-time application of a Finite Impulse Response (FIR) filter to a sensor signal.

2. Server Application

Role: The server's sole responsibility is to act as a data provider. It reads raw signal data from local text files and reliably transmits them to a single connected client.

Workflow:

1. **Initialization:** The server is launched. The user selects a folder on their local machine containing the data files.
2. **Data Selection:** The user specifies whether to transmit a single, specific file or all `.txt` files within the selected folder.
3. **Awaiting Connection:** The server opens a TCP socket and waits for a client to connect.
4. **Configuration Transmission:** Once a client connects, the server immediately sends an initial configuration block. This block contains key parameters, most importantly the `INTERVAL` (sampling period in milliseconds) at which the original data was captured.
5. **File Transmission:** The server proceeds to send the selected file(s) one by one. It uses a specific network protocol to ensure the client can correctly parse the data stream.
6. **Termination:** After the last file is sent, the server sends a special `END_OF_TRANSMISSION` message to signal completion and then closes the connection.

Network Protocol (Critical Point): To ensure data is received without corruption

over a TCP stream, the server uses a **length-prefixing protocol**. Before sending variable-length data (like a filename or its content), it first sends a fixed-size integer that tells the client exactly how many bytes to read next.

The sequence for each file is:

1. `[FILENAME_LENGTH (4 bytes)] [FILENAME (variable length)]`
2. `[CONTENT_LENGTH (8 bytes)] [FILE_CONTENT (variable length)]`

This protocol prevents issues with data buffering and ensures the client can perfectly reconstruct each file it receives.

3. Client Application

Role: The client is the core of the system where all processing, analysis, and visualization occurs. It is designed to be highly responsive and provides immediate visual feedback.

Core Technologies:

- **`cmsisdsp`**: The Python wrapper for the Arm CMSIS-DSP library, used for the high-performance FIR filter.
- **`numpy`** & **`scipy`**: Used for numerical calculations and for designing the FIR filter coefficients.
- **`matplotlib`**: Used for creating and updating the live data plots.
- **`threading`**: Used to separate network operations from the main application logic, ensuring a non-blocking, responsive user interface.

System Architecture (Critical Point): The client utilizes a **multi-threaded architecture** for smooth operation:

- **Network Thread (`receive_data_loop`):** This background thread is dedicated to network communication. It connects to the server, listens for data according to the defined protocol, and places complete data files into a thread-safe `queue`. By handling blocking network calls, it allows the main application to remain fully responsive.
- **Main Thread:** This is the primary thread of execution. It manages the application's lifecycle, retrieves complete data files from the queue, orchestrates all analysis and processing steps, and handles the rendering and updating of the `matplotlib` plot window.

Processing and Analysis Workflow: Once the main thread retrieves a data file from

the queue, it performs a two-stage process: a one-time analysis of the entire file, followed by a real-time simulation.

Stage 1: One-Time Amplitude Analysis Before any plotting occurs, the client performs a quick analysis on the *entire* dataset to calculate its dynamic range.

1. **Raw Signal Amplitude:** The raw ADC values for the entire file are converted to physical units (weights). The peak-to-peak amplitude is calculated as `(maximum weight) - (minimum weight)`.
2. **Filtered Signal Amplitude:** The entire raw signal is filtered at once using the `fir_filter` function. The peak-to-peak amplitude of this complete filtered signal is then calculated.
3. **Result Saving (`save_amplitude_results`):** These two amplitude values are saved to a new text file in the `analysis_results` folder. The output file is named based on the original input filename (e.g., `analysis_FILENAME.txt`), keeping the results neatly organized.

Stage 2: Live Plotting Simulation (`process_and_plot_live_data`) After the analysis is saved, the client begins a real-time visualization of the filtering process.

1. **Initialization:** A `matplotlib` window is created with two subplots: one for the "Raw Data" and one for the "FIR-Filtered Data".
2. **Sample-by-Sample Loop:** The application iterates through the received data points one at a time, pausing for the specified sampling interval between each point to simulate a live data stream.
3. **Pre-Processing (`remove_dc_offset_temp`):** For each small window of data being processed, the DC offset (its average value) is removed. **This is a critical step.** FIR filters are most effective at shaping the AC components (the variations) of a signal. Removing the large DC component first allows the filter to work as designed.
4. **FIR Filtering with CMSIS-DSP (`fir_filter`):**
    - **Coefficient Design:** `scipy.signal.firwin` is used to design the filter coefficients based on a pre-defined cutoff frequency (`FIR_CUTOFF_HZ`).
    - **CMSIS-DSP Execution:** The filtering itself is performed by the `cmsisdsp` library functions. This involves creating a filter "instance," initializing it with the coefficients and a **state buffer** (which acts as the filter's memory of previous samples), and finally calling the `arm_fir_f32` function to process the data window.
5. **Post-Processing:** The DC offset calculated in step 3 is added back to the

filtered result to restore it to its correct absolute scale. The final raw and filtered values are then converted to physical units (weights).

6. **Plot Update (`update_live_plot`):** The new raw and filtered data points are appended to the plot lines, and the `matplotlib` canvas is redrawn to provide the animation.

4. Usage Summary

1. Launch the **server application**.
2. Use the server's interface to select a folder containing data files.
3. Choose to transmit a single file or all files. The server will now be listening for a connection.
4. Run the **client application script** (`main_client.py` or similar).
5. The client will automatically connect, receive configuration, and begin processing the first file.
6. The `analysis_results` folder will be created, and an analysis file will be saved.
7. A `matplotlib` window will open, displaying the raw and filtered signals being plotted in real-time.
8. After one file simulation finishes, the client will automatically start processing the next file from the server, if available.
9. To stop the application, simply **close the plot window**. This will trigger a graceful shutdown of all threads.

5. Client Application Output Explained (Terminal Log)

When you run the client script, you will see a series of messages printed to the terminal. This section explains the meaning of each line in a typical successful run.

Sample Output Log:

1. [CLIENT] CMSIS-DSP library v1.10.1 loaded successfully.
2. [CLIENT] Network thread started.
3. [CLIENT] Main thread waiting for data...
4. [CLIENT] Connected to server.
5. [CLIENT] Received config: INTERVAL:20\nMODE:freq\n
6. [CLIENT] Set interval from server: 20 ms
7. [CLIENT] Calculating signal amplitudes...
8. [CLIENT] Raw data amplitude (peak-to-peak): 22.45
9. [CLIENT] Filtered data amplitude (peak-to-peak): 21.98
10. [CLIENT] Created output directory: analysis_results

11. [CLIENT] Amplitude results saved to: analysis_results\analysis_535g20250502pm427ms20hz50.txt
12. [CLIENT] Starting live simulation for 535g20250502pm427ms20hz50.txt...
13. [CLIENT] Plot initialized for: 535g20250502pm427ms20hz50.txt. Close the plot window to exit.
14. [CLIENT] Received END_OF_TRANSMISSION signal.
15. [CLIENT] Network thread finished.
16. [CLIENT] Finished simulation for 535g20250502pm427ms20hz50.txt.
17. [CLIENT] All files received and processed.
18. [CLIENT] Cleaning up...
19. [CLIENT] Displaying final plot. Close window to exit.
20. [CLIENT] Application exited.

6. Visual Output Explanation (Plots)

The primary output of the client application is the live graphical display. When a file is processed, a `matplotlib` window opens, containing two plots arranged vertically. This window provides an immediate, intuitive understanding of the filter's effect on the signal.

Example Plot Output:

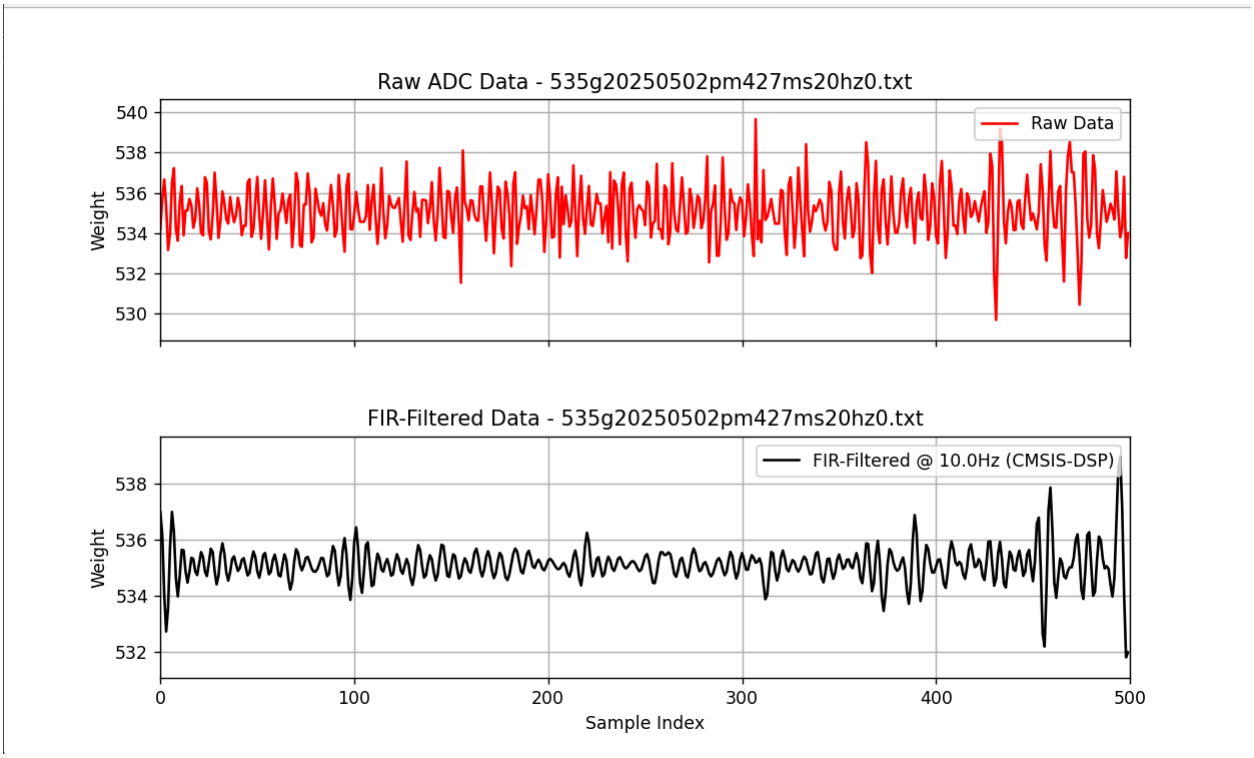
Top Plot: Raw ADC Data

- **Content:** This plot displays the incoming sensor data *after* it has been converted to physical units (Weight) but *before* any filtering has been applied.
- **X-Axis:** Represents the sample index, showing the progression of the signal over time.
- **Y-Axis:** Represents the measured weight.
- **Interpretation:** This is the baseline signal. It contains the fundamental oscillation as well as any high-frequency noise or jitter captured by the sensor. You can observe the original, unfiltered dynamic range and characteristics here.

Bottom Plot: FIR-Filtered Data

- **Content:** This plot displays the signal *after* it has been processed by the CMSIS-DSP FIR filter.
- **X-Axis:** Aligned with the top plot, representing the same sample index.

- **Y-Axis:** Represents the measured weight after filtering.
- **Interpretation:** This is the result of the DSP operation. The key difference you should observe is that this signal is significantly "smoother" than the raw data. The sharp, jagged edges (high-frequency noise) have been removed by the low-pass filter, leaving behind the core, slower-moving oscillation. The plot legend, `FIR-Filtered @ 10.0Hz`, indicates the cutoff frequency used for the filter, meaning frequencies above 10 Hz have been attenuated. The peak-to-peak amplitude (Y-axis range) of this signal will typically be slightly lower than the raw signal, as some signal energy is always removed during filtering.



Analysis for file: 535g20250502pm427ms20hz0.txt

Analysis timestamp: 2025-06-25 10:37:07

Raw Data Peak-to-Peak Amplitude: 19.5871

Filtered Data Peak-to-Peak Amplitude: 11.7462