This document provides a detailed, function-by-function explanation of the final, working version of the Standalone FIR Filtering Application.

Table of Contents

1. Global Variables & Configuration

- **USER_SELECTED_INTERVAL_MS**: Stores the current plotting speed (in milliseconds) selected by the user via the GUI. This value is updated in real-time.
- **g_animation**: Holds the active FuncAnimation object. This is critical for starting, stopping, and modifying the animation.
- **g_all_raw_weights, g_all_filtered_weights**: NumPy arrays that store the *complete* processed data for the currently selected file. The animation function reads from these arrays to draw each frame.
- **PLOT_WINDOW_SIZE**: A constant that defines the width (in samples) of the scrolling plot window, ensuring a close-up view of the data.
- **FIR_CUTOFF_HZ**: The fixed cutoff frequency for the low-pass FIR filter. All frequencies in the signal above this value will be attenuated.
- **current_fig, current_axes, etc.**: Global references to the Matplotlib figure and plot elements, allowing any function to access and modify them.

2. Core DSP and Analysis Functions

calculate_and_process_data(raw_adc_values, file_name)

- **Purpose**: To perform all the heavy, one-time calculations for an entire data file. It processes the full signal to get the final data arrays ready for animation.
- **Working Process**:
  1. Calculates the **stable DC offset** (the average) of the entire raw signal.
  2. Converts the raw ADC values to physical weights and calculates the **peak-to-peak amplitude** of the raw signal.
  3. Subtracts the stable DC offset from the entire signal.
  4. Passes the full, DC-removed signal to the fir_filter function.
  5. Adds the stable DC offset back to the filtered result to create the final reconstructed signal.
  6. Calculates the peak-to-peak amplitude of this final filtered signal.
  7. Calls save_amplitude_results to write the analysis to a file.
  8. Returns two complete NumPy arrays: one for the raw weights and one for the filtered weights.

fir_filter(values, cut_off_frequency, sampling_rate)

- **Purpose**: To apply the low-pass FIR filter using the cmsisdsp library.
- **Working Process**:
  1. **Design Coefficients**: Uses scipy.signal.firwin to calculate the filter's mathematical coefficients.
  2. **Prepare CMSIS-DSP**: Creates the necessary data structures (fir_instance and state_f32 buffer) required by the C-based library.
  3. **Initialize**: Calls dsp.arm_fir_init_f32() to configure the filter with the coefficients and state buffer.
  4. **Execute**: Calls dsp.arm_fir_f32() to perform the high-speed filtering operation on the input data.
  5. Returns the resulting filtered data array.

normalize_to_weights(values)

- **Purpose**: To convert abstract integer ADC values into meaningful physical units (grams).
- **Working Process**: Applies a linear formula ((value / scale_factor) - offset) to each data point using pre-defined calibration constants.

3. File and Output Management

save_amplitude_results(...)

- **Purpose**: To save the analysis results to a permanent text file.
- **Working Process**:

1. Creates an analysis_results folder if it doesn't exist.
2. Creates a new file named after the original input file (e.g., analysis_datafile.txt).
3. Writes the calculated raw and filtered peak-to-peak amplitudes into the file, along with a timestamp.

4. GUI and Plotting Functions

init_plot(file_list, data_queue)

- **Purpose**: To create the main application window, including the plots and all interactive widgets, at startup.
- **Working Process**:
  1. Creates the Matplotlib figure and two empty subplots.
  2. Adjusts the layout to make space for the widgets on the left.
  3. Creates and configures the **Speed Control** (RadioButtons) widget. It defines and attaches the on_speed_change callback function to it.
  4. Creates and configures the **File Selection** (RadioButtons) widget, populating it with the list of .txt files found in the user's selected directory. It defines and attaches the on_file_change callback function to it.

start_new_animation(all_raw, all_filtered, file_name)

- **Purpose**: To reset the plot and begin a new animation for a newly selected file.
- **Working Process**:
  1. **Stop Previous Animation**: It first checks if another animation (g_animation) is running and explicitly stops it. This is critical for preventing overlapping plots.
  2. **Update Global Data**: It assigns the newly processed data arrays (all_raw, all_filtered) to the global variables so the animation function can access them.
  3. **Reset Plot**: It clears the plot titles and resets the X and Y axis limits based on the new data's range, ensuring the plot is framed correctly.
  4. **Create FuncAnimation**: It creates a new FuncAnimation object. This is the core of the modern plotting architecture. It tells Matplotlib to repeatedly call the update_animation_frame function at an interval determined by USER_SELECTED_INTERVAL_MS.
  5. The animation starts automatically.

update_animation_frame(frame)

- **Purpose**: This function is the engine of the animation. It is called automatically by FuncAnimation for every single frame.
- **Working Process**:
  1. The frame argument represents the current sample number (e.g., 0, 1, 2, ...).
  2. It updates the data for the two plot lines, showing all points from the beginning

up to the current frame.
3. It implements the **scrolling window** by adjusting the X-axis limits to show only the last PLOT_WINDOW_SIZE samples once the plot fills up.
4. When the final frame is reached, it sets the g_simulation_running flag to False, indicating that the application is ready to process another file.

5. User Interaction (Callbacks)

on_file_change(label, data_queue)

- **Purpose**: This function is executed automatically whenever the user clicks on a filename in the GUI.
- **Working Process**:
  1. It first stops any animation that is currently running.
  2. It constructs the full path to the newly selected file.
  3. It launches a new **background thread** that runs the load_and_process_file_worker function. This prevents the GUI from freezing while the (potentially large) file is being read and processed.

on_speed_change(label) (defined inside init_plot)

- **Purpose**: This function is executed automatically whenever the user clicks on a new speed option.
- **Working Process**:
  1. It updates the global USER_SELECTED_INTERVAL_MS variable.
  2. It directly modifies the interval property of the currently running g_animation object, causing the plotting speed to change instantly and smoothly.

6. Threading and Data Handling

load_and_process_file_worker(filepath, data_queue)

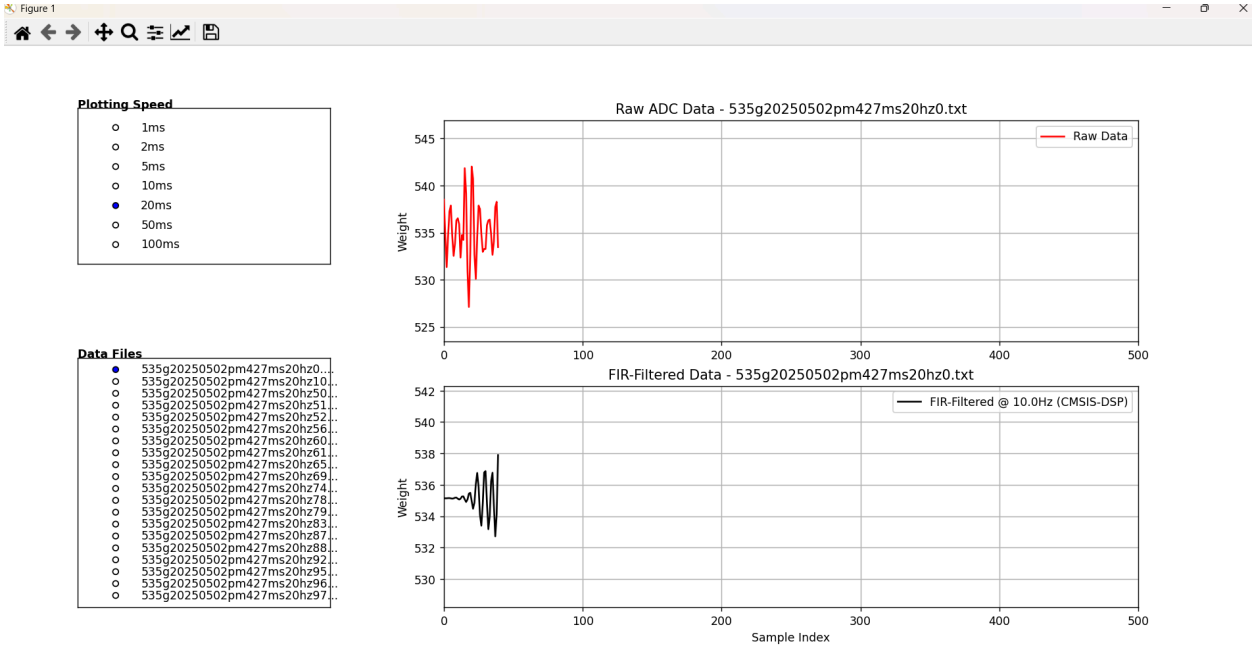
- **Purpose**: To perform all file reading and heavy data processing in a background thread, keeping the GUI responsive.
- **Working Process**:
  1. It opens and reads the selected text file.
  2. It calls calculate_and_process_data to perform all the analysis and filtering.
  3. Once the processing is complete, it places the final, ready-to-plot data arrays into the data_queue for the main thread to retrieve.

7. Main Execution Block

if __name__ == "__main__"

- **Purpose**: The main entry point of the application.
- **Working Process**:

1. **Select Directory**: It first opens a native file dialog, prompting the user to select the directory where their data files are located.
2. **Scan for Files**: It finds all .txt files in that directory.
3. **Initialize Plot**: It calls init_plot, passing it the list of found files to populate the GUI.
4. **Start GUI Timer**: It creates and starts a fig.canvas.new_timer(). This is a non-blocking timer that runs in the main GUI thread. Every 100 milliseconds, it calls the check_queue function.
5. **check_queue()**: This small helper function safely checks if there is new, fully processed data in the data_queue. If there is, it retrieves the data and calls start_new_animation to begin the plotting.
6. **plt.show()**: This is the final and most important call. It hands over full control to Matplotlib's event loop, which draws the window and manages all user interactions (button clicks, window resizing, etc.) without freezing. The program will stay on this line until the user closes the plot window.



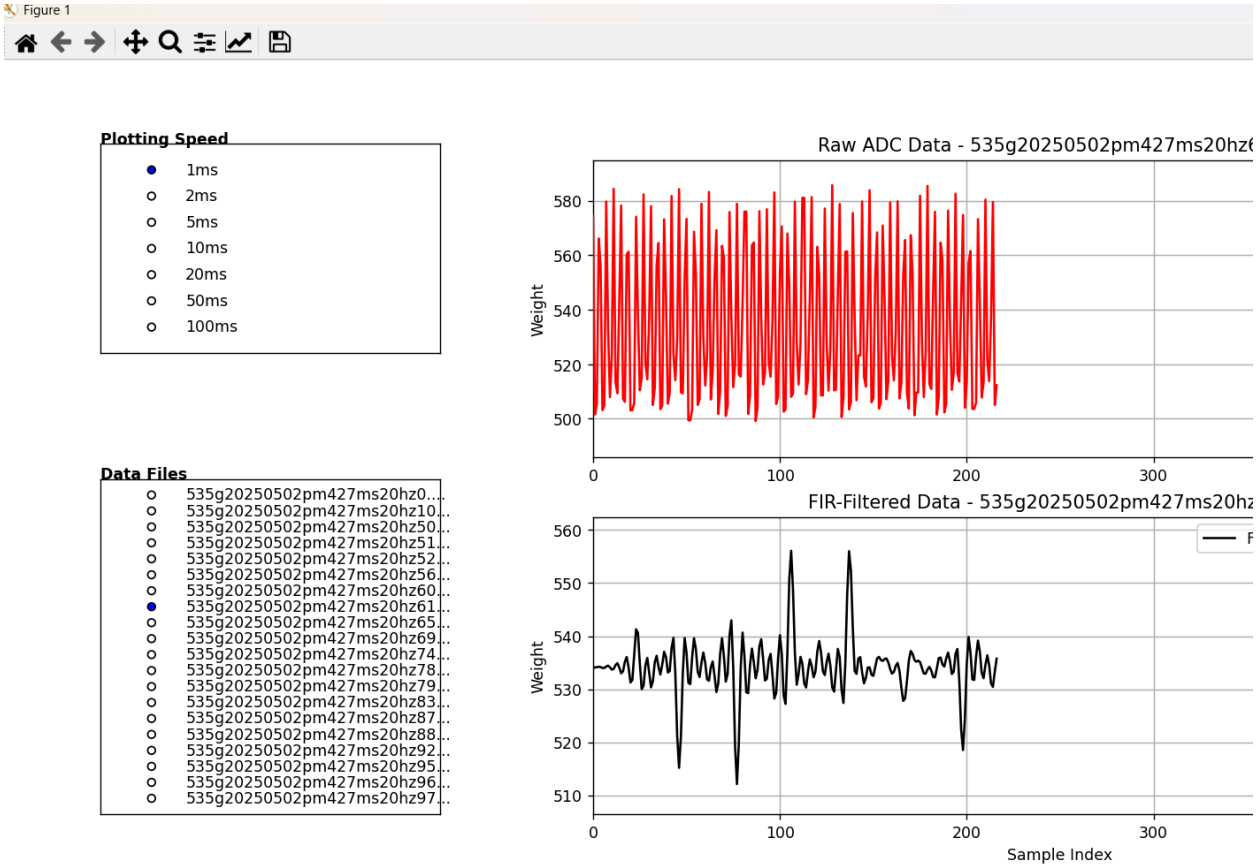


Figure 1
Plotting Speed
1ms
2ms
5ms
10ms
20ms
50ms
100ms
Data Files
535g20250502pm427ms20hz0...
535g20250502pm427ms20hz10...
535g20250502pm427ms20hz50...
535g20250502pm427ms20hz51...
535g20250502pm427ms20hz52...
535g20250502pm427ms20hz56...
535g20250502pm427ms20hz60...
535g20250502pm427ms20hz61...
535g20250502pm427ms20hz65...
535g20250502pm427ms20hz69...
535g20250502pm427ms20hz74...
535g20250502pm427ms20hz78...
535g20250502pm427ms20hz79...
535g20250502pm427ms20hz83...
535g20250502pm427ms20hz87...
535g20250502pm427ms20hz88...
535g20250502pm427ms20hz92...
535g20250502pm427ms20hz95...
535g20250502pm427ms20hz96...
535g20250502pm427ms20hz97...
Raw ADC Data - 535g20250502pm427ms20hz
Weight
580
560
540
520
500
0
100
200
300
FIR-Filtered Data - 535g20250502pm427ms20hz
Weight
560
550
540
530
520
510
0
100
200
300
Sample Index