ADC System with RF Frontend Simulation and FIR Filter Design

1. Project Overview

Title ADC System with RF Frontend Simulation and FIR Filter Design

Introduction:
Digital filters are fundamental components in signal processing, used to remove unwanted frequencies or extract useful signal bands. Among them, Finite Impulse Response (FIR) filters are widely preferred due to their inherent stability and linear phase properties. FIR filters find applications in communication systems, biomedical devices, and digital instrumentation.

This project focuses on designing and implementing a bandpass FIR filter with a passband of 90–110 kHz at a sampling rate of 1 MHz. The filter is designed using Python (with the Remez algorithm) and implemented in Verilog RTL. Its performance is verified through simulation in ModelSim, with additional validation carried out using Multisim circuit simulation. The workflow demonstrates both DSP theory and hardware implementation, ensuring correctness by comparing Python and Verilog outputs.

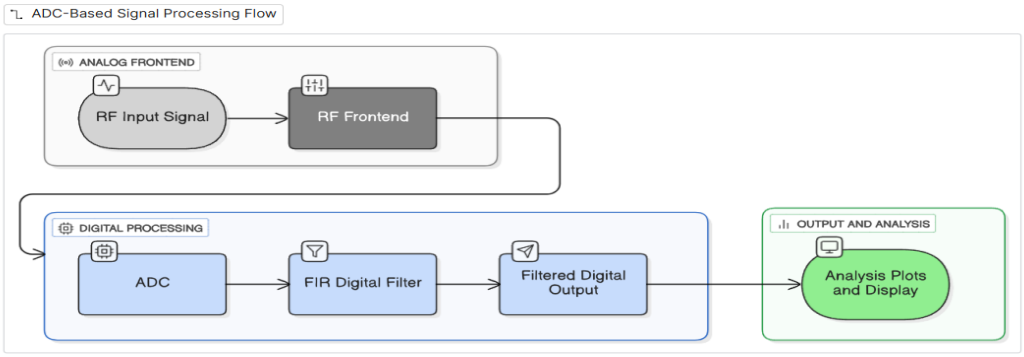
Description:
The project involves designing, implementing, and verifying a Finite Impulse Response (FIR) bandpass filter using Verilog RTL. The filter targets a 90–110 kHz passband with a sampling rate of 1 MHz, designed in Python and verified against a Verilog model. The implementation uses fixed-point arithmetic (Q1.23 format) and is validated through simulation, comparison with a Python golden model, and frequency/time-domain analysis.

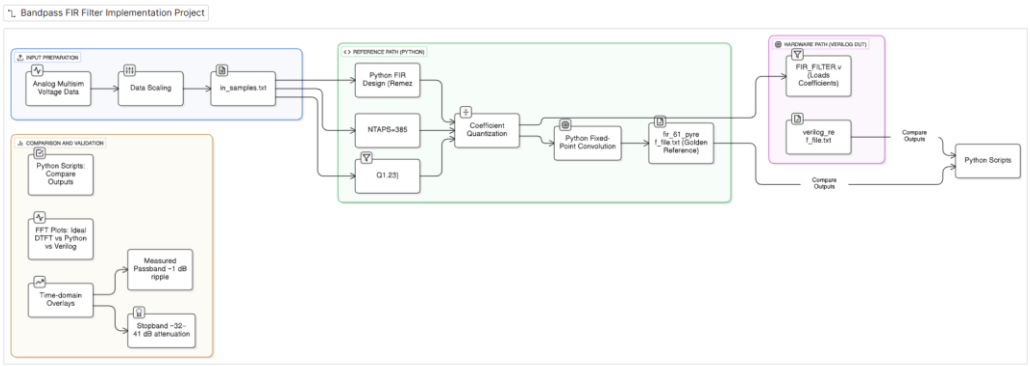
Objective:
To design and verify a parameterized FIR filter that achieves the desired passband and stopband specifications, ensuring correctness of the Verilog implementation through Python cross-verification.

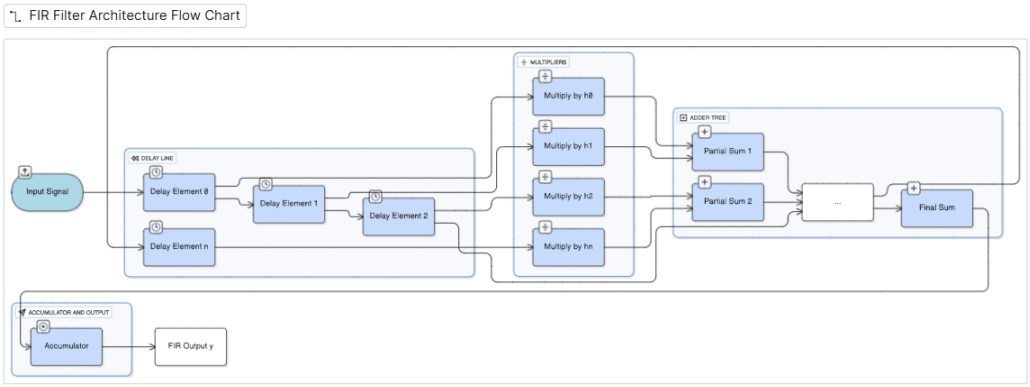
Key Features:

- Bandpass FIR filter (90–110 kHz at fs = 1 MHz).
- Parameterized number of taps (default = 385).
- Coefficient quantization in Q1.23 fixed-point format.
- Verilog RTL implementation using direct convolution.
- Python golden model for reference output.
- Testbench for simulation in ModelSim.
- Automated Python–Verilog result comparison.

2. Architecture Diagram







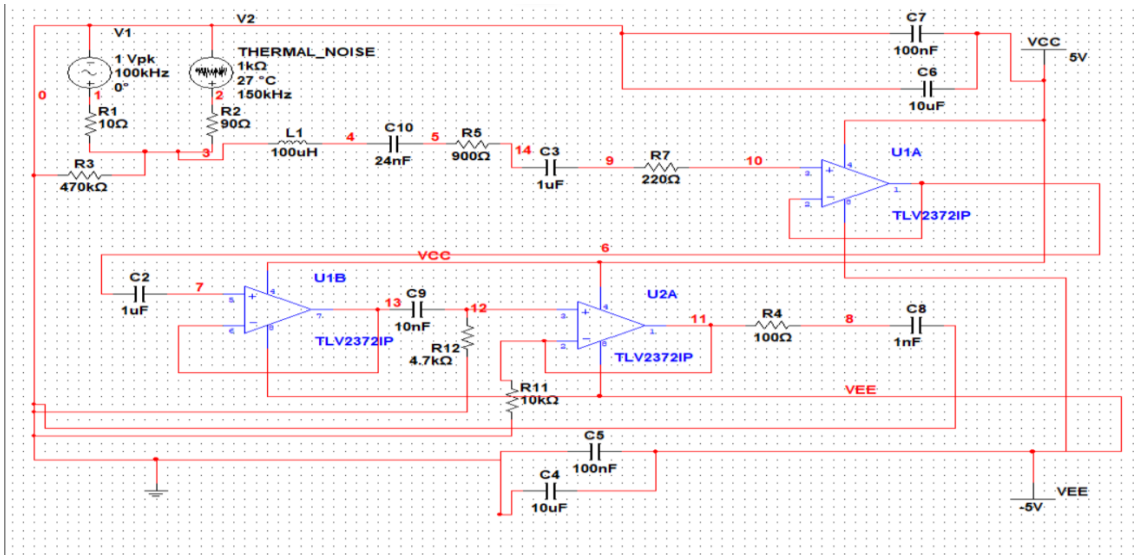
FIR Filter Architecture Flow Chart — showing delay line, multipliers, adder tree, and accumulator for final output.

Data Flow:

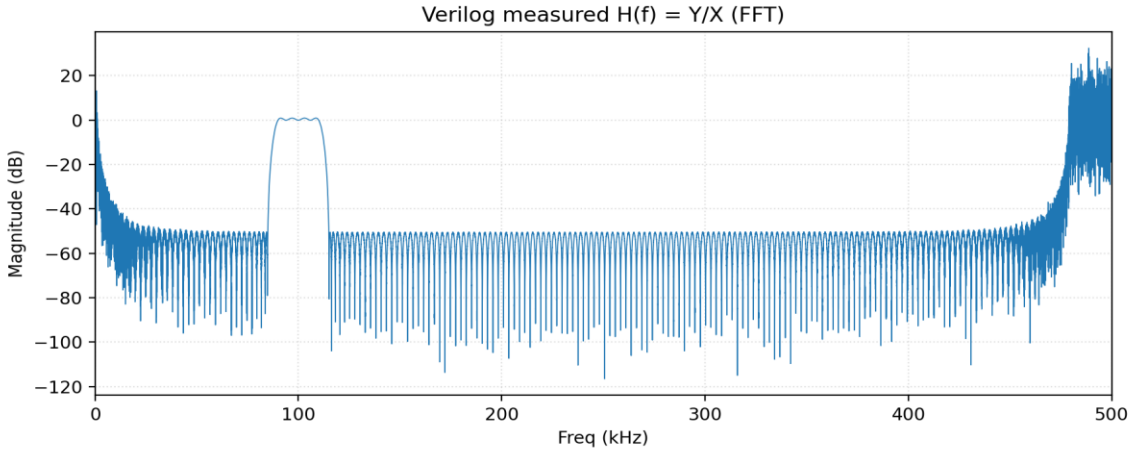
1. Input samples are read and passed through a shift register delay line.
2. Coefficients are multiplied with delayed samples (MAC operation).
3. Accumulator sums the products, rounding and clipping ensure valid output range.
4. Testbench compares Verilog output with Python reference.

3. Filter Design Specifications

- Sampling frequency (fs) = 1 MHz
- Passband = 90–110 kHz
- Stopband attenuation ≈ 30 dB
- Passband ripple ≈ 3 dB
- Number of taps = 385
- Quantization: 23 fractional bits
- Input samples were derived from Multisim simulations. Voltage values were scaled appropriately before being used as in_samples.txt for Python and Verilog verification.



RF frontend circuit – Multisim



FFT response of designed FIR filter.

4. Module Descriptions

Module	Description	Inputs / Outputs
FIR_FILTER.v	Implements FIR convolution using delay line, multipliers, accumulator, rounding & clipping.	clk, rst, sample_in, sample_valid → sample_out, out_valid
FIR_TB.v	Testbench that loads coefficients, drives input samples, and logs outputs for comparison.	File I/O based input/output handling.
fir.py	Python script to design filter, quantize coefficients, generate reference outputs.	Generates coeffs, reports, debug files.
ver_py_check_fixed.py	Compares Python and Verilog outputs, aligns streams, reports mismatches.	Outputs match/mismatch summary.
plot_compare.py	Generates time-domain and frequency-domain plots for verification.	Produces PNG plots.

5. Testbench & Simulation

- Clock: 10 ns period (100 MHz).
- Reset initializes DUT and clears registers.
- Coefficients loaded from fir_61_coeffs_hex.txt.
- Input samples read from in_samples.txt.
- Output samples written to verilog_ref_file.txt.
- Debug info written to verilog_dbg.txt.
- Pipeline flushed with (NTAPS-1) zeros to match Python alignment.

6. Results

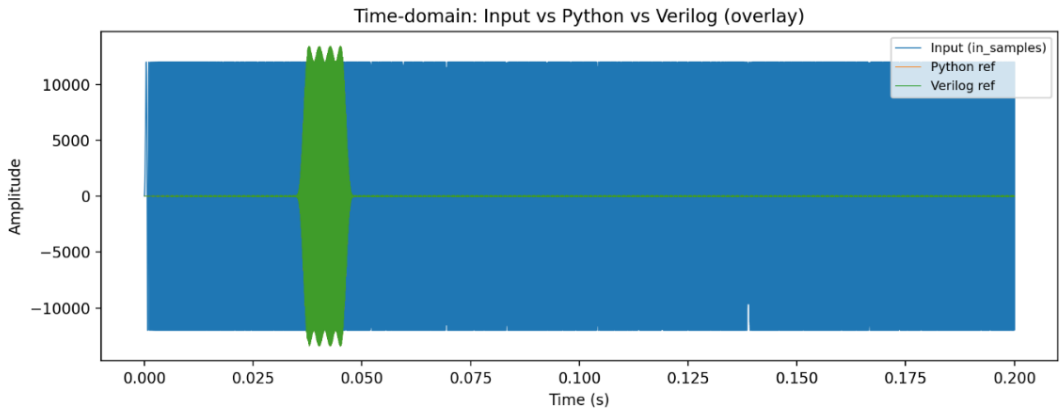
The FIR filter was verified in both time and frequency domains, and Python outputs were compared against Verilog and the ideal DTFT response.

```
C:\FIR>python ver_py_check.py
Loaded Python ref 'fir_61_pyref_file.txt' len = 200000
Loaded Verilog ref 'verilog_ref_file.txt' len = 200384
Loaded Verilog ref 'py_ideal_conv.txt' len = 200000
Length mismatch: Verilog longer by 384 samples. Trimming first 384 samples from Verilog ref.
Comparing 200000 samples after trimming (if any).

First samples (expected=python | verilog | Ideal):
0000:           16  |           16  |           16
0001:           16  |           16  |           16
0002:           16  |           16  |           16
0003:           16  |           16  |           16
0004:           16  |           16  |           16
0005:           16  |           16  |           16
0006:           16  |           16  |           16
0007:           16  |           16  |           16
0008:           16  |           16  |           16
0009:           16  |           16  |           16
0010:           16  |           16  |           16
0011:           16  |           16  |           16
0012:           16  |           16  |           16
0013:           16  |           16  |           16
0014:           16  |           16  |           16
0015:           16  |           16  |           16

No mismatches found. Files match for all compared samples ✅
Python sum=5721  Verilog sum=5721  Ideal sum=5721
```

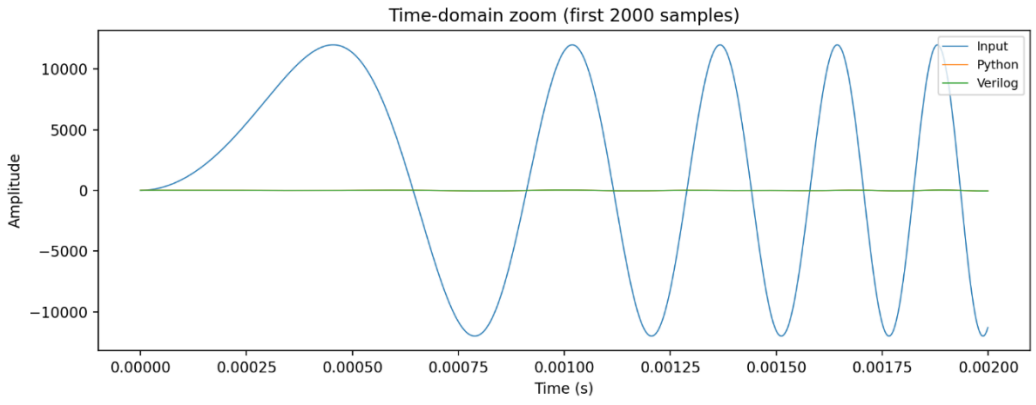
```
C:\FIR>python plot_compare.py
Loading files...
Lengths: input=200000 python=200000 verilog=200384 coeffs=385
After simple alignment: py2_len=200000 ver2_len=200000
Aligned lengths -> L = 199998
Wrote time_overlay.png
Wrote time_zoom_overlay.png
Wrote py_fft.png
Wrote ver_fft.png
Wrote fft_overlay.png
Wrote ideal_vs_measured.png

Passband/Stopband summary (center @ 100 kHz):
Python-measured: center=-0.068 dB, min=-0.078 dB, max=0.910 dB, ripple=0.988 dB
Verilog-measured: center=-0.068 dB, min=-0.078 dB, max=0.910 dB, ripple=0.988 dB
Ideal DTFT: center=0.000 dB, min=-0.000 dB, max=0.971 dB, ripple=0.971 dB
Python stopband H(f) = 32.353 dB (attenuation = -32.353 dB)

All plots written. Inspect PNGs: time_overlay.png, time_zoom_overlay.png, fft_overlay.png, ideal_vs_measured.png
```

6.1 Time-Domain Verification

- **Python and Verilog outputs overlap perfectly.**



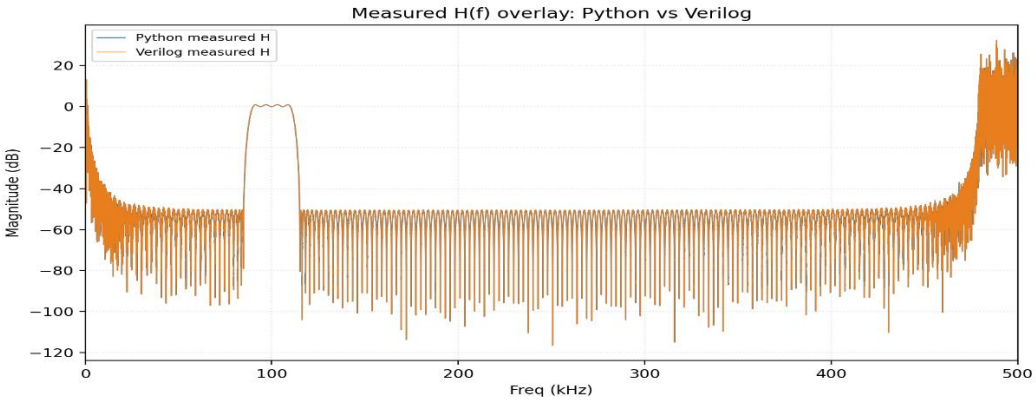
time_overlay.png – Time-domain overlay of input, Python, and Verilog outputs.



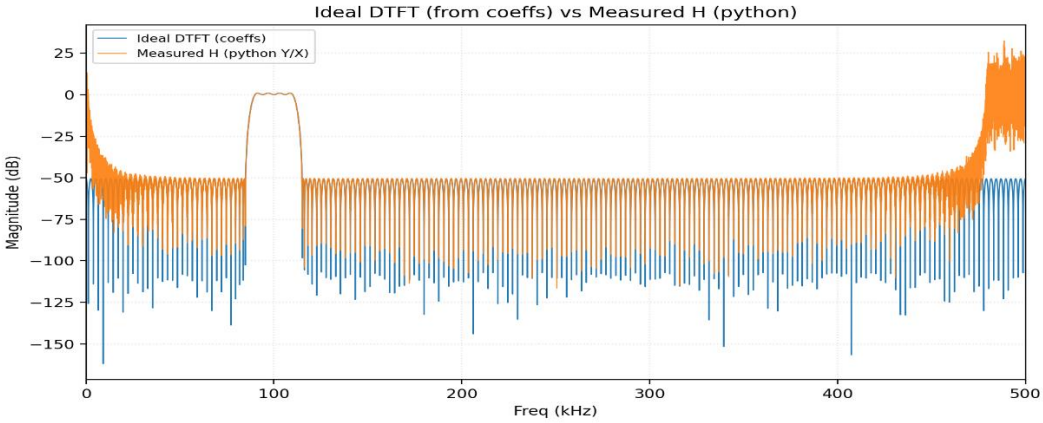
time_zoom_overlay.png – First 2000 samples zoomed.

6.2 Frequency-Domain Verification

- FFT analysis confirms bandpass response centered at 100 kHz.



fft_overlay.png – Frequency-domain comparison of Python vs Verilog response.



ideal_vs_measured.png – Ideal DTFT vs measured FIR filter response.

6.3 Report Summary (from fir_61_report.txt)

- NTAPS=385
- QT=23
- Floating_pre_center_db=-0.498487
- Measured_center_db=-0.068479
- pass_min=-0.078604
- pass_max=0.909481
- ripple=0.988085
- worst_stop=41.000926

7. Challenges Faced

- **Coefficient Quantization:**
  Converting floating-point filter coefficients into Q1.23 fixed-point format introduced quantization noise, increasing ripple compared to the ideal design.
- **Shift Register Updates:**
  Initially, using non-blocking assignments in the delay line caused misalignment in MAC operations. This was fixed by switching to blocking assignments.
- **Output Alignment:**
  Python reference produced full-length convolution results, while Verilog required pipeline flushing. Careful handling ensured both outputs aligned correctly.
- **Verification Effort:**
  Debugging mismatches required creating automated comparison scripts (ver_py_check_fixed.py) to catch even single-sample differences.

8. How to Run

1. Design filter (Python): python fir.py.
2. Generates coefficients, reference outputs, and reports.
3. Simulate Verilog:

   vlog FIR_FILTER.v FIR_TB.v

   vsim -c work.FIR_TB -do "run -all; quit"

Produces verilog_ref_file.txt and verilog_dbg.txt.

4. Compare outputs:

   --python ver_py_check_fixed.py --py fir_61_pyref_file.txt --ver verilog_ref_file.txt

Generate plots: python plot_compare.py Produces PNGs: time_overlay.png, fft_overlay.png, ideal_vs_measured.png, etc.

9. Future Improvements

- Optimization for FPGA:
  Current design uses direct convolution (O(NTAPS)). Future versions could use fast FIR architectures (like distributed arithmetic or FFT-based filtering) for efficiency.
- Pipelining:
  Introducing pipeline stages in the MAC unit would allow higher operating frequency and reduce critical path delay.
- Configurable Filter Types:
  Extend the Python/Verilog framework to generate low-pass, high-pass, or multi-band FIR filters with minimal changes.

- Coefficient Reload Feature:
  Add logic in Verilog to allow coefficient reloading at runtime, enabling adaptive filtering.
- Higher Stopband Attenuation:
  By increasing the number of taps or using more advanced design methods, stopband attenuation can be improved beyond –30 dB.

10. Conclusion

This project successfully demonstrated the end-to-end design and verification of a FIR bandpass filter. The filter was designed in Python, quantized into fixed-point coefficients, and implemented in Verilog RTL. Simulation results confirmed a close match between Python and Verilog outputs, both in time and frequency domains. The measured passband ripple and stopband attenuation were within acceptable limits, validating the correctness of the design.

Additionally, Multisim simulations provided circuit-level validation, bridging the gap between theoretical design and practical implementation. The project highlights the importance of cross-domain verification (Python, Verilog, and Multisim) in ensuring accurate digital filter design, making the methodology suitable for future FPGA or ASIC implementation.

11. References

- Python 3.10, NumPy, SciPy (for filter design & simulation)
- ModelSim (for Verilog simulation)
- Verilog HDL standard documentation