Using Python for Analysis and Verification of Mixed-mode Signal Chains for Analog Signal Acquisition

Mark Thoren and Cristina Șuteu

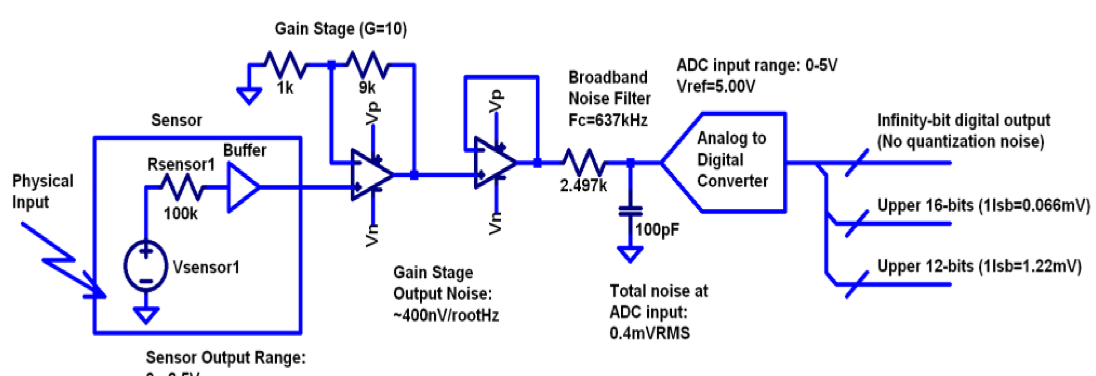
Analog Devices, Inc, System Development Group

Introduction

Mixed mode signal chains are, in a nutshell, systems that transduce a real-world signal to an electrical signal, and then digitize it. At any point in this chain, the signal is subject to distortion or additive noise, from various sources such as component tolerances, temperature drift, interference from adjacent signals and supply voltage variations. Many of the imperfections in the signal chain can be compensated for. However, noise is the only one that cannot. Thus, this work details the analysis and validation of mixed-mode signal chains., focusing on noise. Signal chain elements will be verified using Python to drive low-cost instrumentation and evaluation boards via the Linux IIO framework.

A Generic Signal Chain

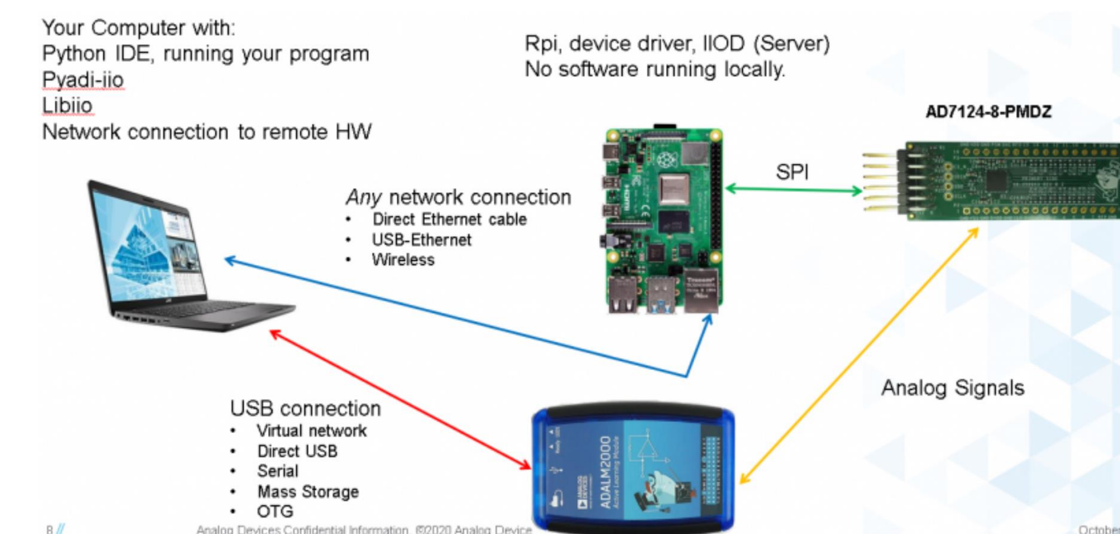
Most analog signals eventually find their way into the digital domain via an Analog to Digital converter (ADC). A generalized signal chain is shown below, with noise sources labeled. While ADC architectures vary considerably, converters for precision applications tend to have 1) internal digital filtering, that limits noise bandwidth, and 2) a quantization noise that is lower than the ADC's own thermal noise.



The Set-up

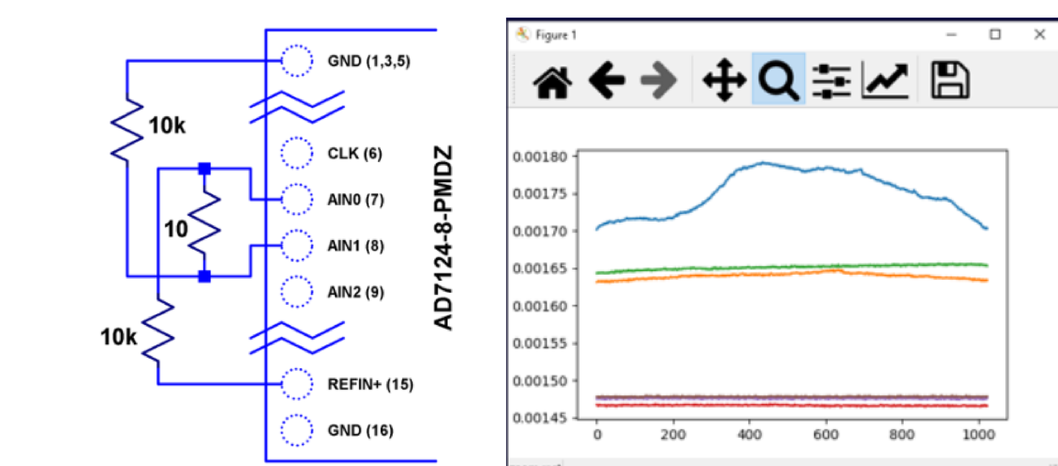
To achieve the results presented here, the experimental setup shown below was used. This is made up of the following elements:

- AD7124-8 -8-Channel, Low Noise, Low Power, 24-Bit, Sigma-Delta ADC
- Raspberry Pi 4 running a kernel with AD7124 device driver support
- ADALM2000 – multifunction USB test instrument



Measuring and Modeling ADC Noise

The results presented on the plot on the right, were achieved by measuring the voltage of two shorted ADC inputs. The additional circuitry is to impose a 1.25mV signal across the input, which overcomes the15µV uncalibrated offset of the AD7124-8. The "wandering" can be due to a number of factors - the internal reference warming up, the external resistors warming up (and hence drifting), or parasitic thermocouples. The lower traces in the plot are after wrapping the AD7124 and resistor divider in antistatic bubble wrap, then waiting half an hour. Measured noise under these conditions was about 565nV RMS, on par with data sheet specifications.



Modeling ADC Digital Filters

An ADC's digital filter response must be known in order to evaluate the impact of signal chain noise, especially if the noise is not flat. While certain ADCs have "brickwall" filters that can be approximated by a rectangular response, ADCs for instrumentation tend to have cascaded SINC filters. The code listing below shows how to construct a model of the AD7124's 50/60Hz rejection filter, and then verifies the model using scipy.signal.freqz. To its right, the resulting impulse response plot is displayed in comparison with the datasheet equivalent.

```
# Calculate SINC1 oversample ratios for 50, 60Hz
osr50 = int(f0/50)
osr60 = int(f0/60)

# Create "boxcar" SINC1 filters
sinc1_50 = np.ones(osr50)
sinc1_60 = np.ones(osr60)

# Calculate higher order filters
sinc2_50 = np.convolve(sinc1_50, sinc1_50)
sinc3_50 = np.convolve(sinc2_50, sinc1_50)
sinc4_50 = np.convolve(sinc2_50, sinc2_50)

# Here's the filter from datasheet Figure 91,
# SINC4-ish filter with one three zeros at 50Hz, one at 60Hz.
filt_50_60_rej = np.convolve(sinc3_50, sinc1_60)

# Normalize to unity gain by dividing by sum of all taps
sinc1_50 /= np.sum(sinc1_50)
sinc1_60 /= np.sum(sinc1_60)
sinc2_50 /= np.sum(sinc2_50)
sinc3_50 /= np.sum(sinc3_50)
sinc4_50 /= np.sum(sinc4_50)
filt_50_60_rej /= np.sum(filt_50_60_rej)


# freqz: Compute the frequency response of a digital filter.
# Older versions of SicPy return w as radians / sample, newer take an optional
# sample rate argument (fs). Computing frequencies (freqs)
# manually for backwards compatibility.

w, h = signal.freqz(filt_50_60_rej, 1, worN=16385, whole=False) #, fs=f0)
freqs = w * f0/(2.0*np.pi)
hmax = abs(max(h)) #Normalize to unity
response_dB = 20.0 * np.log10(abs(h)/hmax)
```

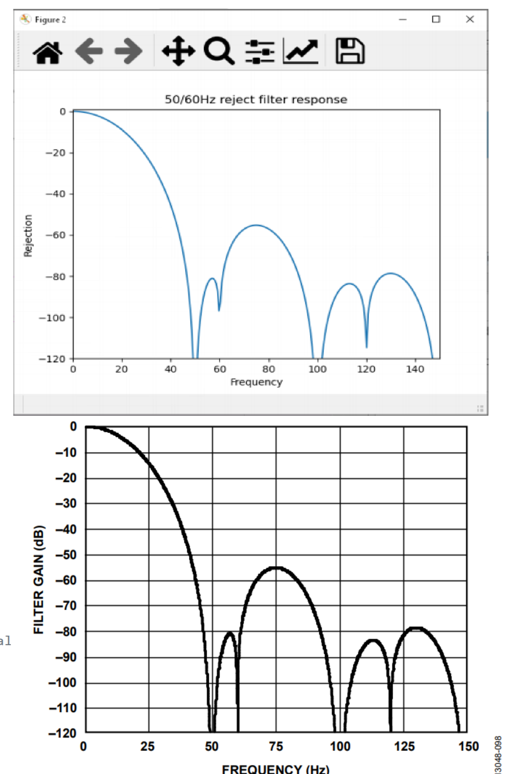
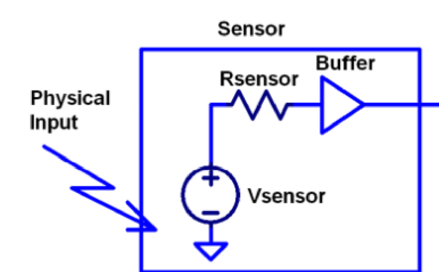


Figure 91. Sinc³ Filter Response (50 SPS Output Data Rate, Zero Latency Disabled or 12.5 SPS Output Data Rate, Zero Latency Enabled, REJ60 = 1)

Resistance is Futile: A Fundamental Sensor Limitation

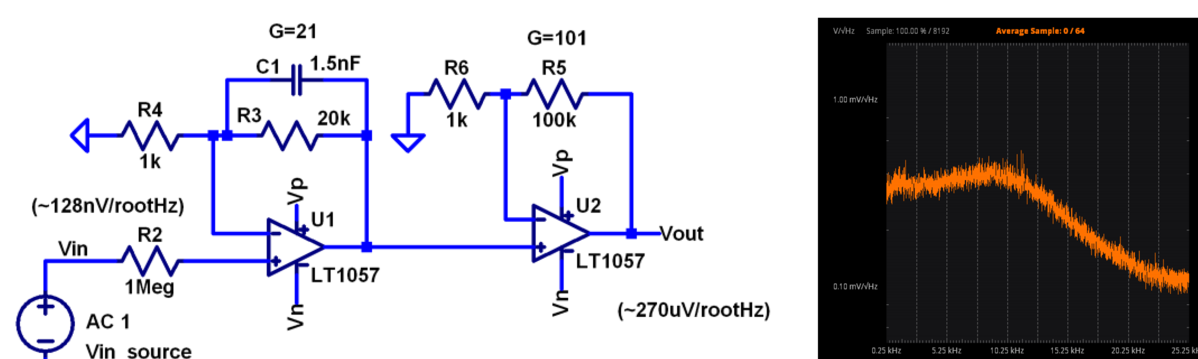
At some point, a sensor with an electrical output will include an element with a finite resistance (or more generally, impedance), represented in the diagram by Rsensor. This is one fundamental noise limit that cannot be improved upon.



Laboratory Noise Source

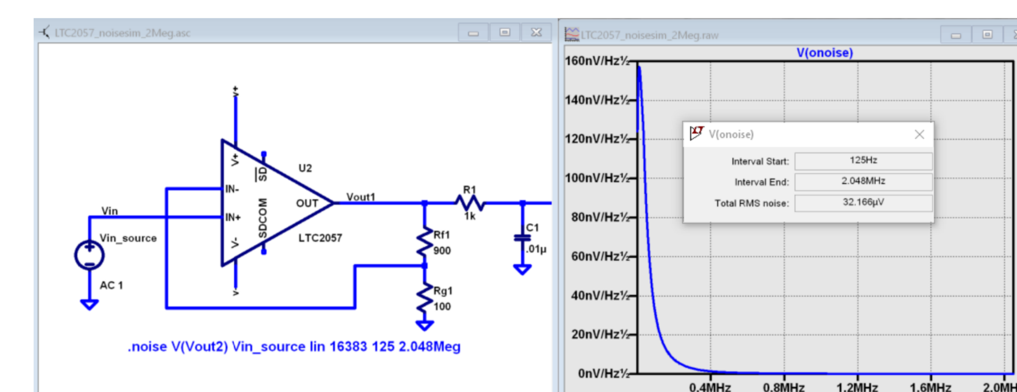
Calibrated noise sources are invaluable for both emulating sensor noise and measuring a signal chain's response to noise. A resistive noise source was constructed as a "reality check" in order to validate data acquisition hardware and analysis code. The circuit used and the results of the measurements can be seen in figure x.

- 1MΩ resistor provides 127nV/√Hz noise density source
- Gain of 2121 produces 0.26mV /√Hz output
- Validated with ADALM2000 and Scopy spectrum analyzer



Modeling Signal Chains In LTspice

The amplifier used in to measure the noise density of the resistor has its own noise limitations. In order to make sure that parts are chosen correctly, the noise spectrum was modeled in Ltspice, as shown in the diagram. The results of the simulation are validated with the Python function listed below the diagram. The two are quite close, returning a value of approximately 32µVRMS.



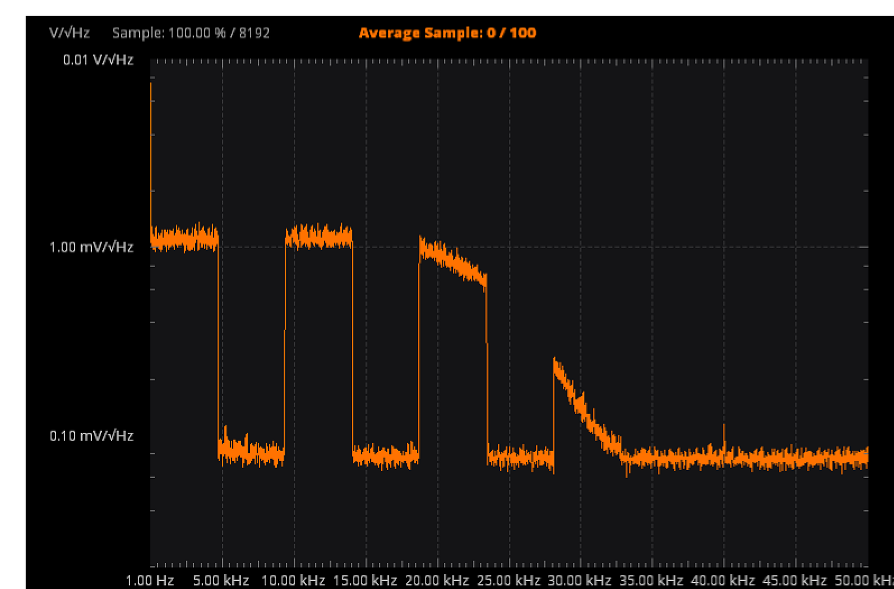

```
# Function to integrate a power-spectral-density
# The last element represents the total integrated noise
def integrate_psd(psd, bw):
    integ_of_psd_squared = np.zeros(len(psd))
    integrated_psd = np.zeros(len(psd))
    integ_of_psd_squared[0] = psd[0]**2.0

    for i in range(1, len(psd)):
        integ_of_psd_squared[i] += integ_of_psd_squared[i-1] + psd[i-1] ** 2
        integrated_psd[i] += integ_of_psd_squared[i]**0.5
    integrated_psd *= bw**0.5
    return integrated_psd
```

Generating Test Noise

Digitally generated noise has the advantage that arbitrary noise spectra can be generated. The code listing below is a function that accepts a desired noise density vector and generates a time series by randomizing the phase of each frequency component, then taking the inverse Fourier transform. The resulting time series can then be sent to an appropriate digital to analog converter or arbitrary waveform generator (AWG) such as an ADALM2000 USB instrument's AWG output. The signals noise density is validated with Scopy's Spectrum Analyzer, as it can be seen in the plot further down.

```
def time_points_from_freq(freq, fs=1, density=False): #DC at element zero,
    N=len(freq)
    rnd_ph_pos = (np.ones(N-1, dtype=np.complex)*
                  np.exp(1j*np.random.uniform(0.0, 2.0*np.pi, N-1)))
    rnd_ph_neg = np.flip(np.conjugate(rnd_ph_pos))
    rnd_ph_full = np.concatenate(([1],rnd_ph_pos,[1], rnd_ph_neg))
    r_spectrum_full = np.concatenate((freq, np.roll(np.flip(freq), 1)))
    r_spectrum_rnd_ph = r_spectrum_full * rnd_ph_full
    r_time_full = np.fft.ifft(r_spectrum_rnd_ph)
#    print("RMS imaginary component: ", np.std(np.imag(r_time_full)),
#          " Should be close to nothing")
    if (density == True):
        r_time_full *= N*np.sqrt(fs/(N)) #Note that this N is "predivided" by 2
    return(np.real(r_time_full))
```



Calculating ADC filter Noise Bandwidth

The effective noise bandwidth (ENBW) of any filter is the bandwidth of a "brick wall" filter that passes the same total noise. The ENBW is calculated by integrating the square of the filter response. The code listing below calculates the ENBW for an arbitrary filter response. Applying this function to the AD7124 SINC4 filter results in an ENBW of 31 Hz.

```
# Equivalent noise bandwidth of an arbitrary filter, given
# frequency response magnitude and bandwidth per point
def arb_enbw(fresp, bw):
    integ_of_fresp_sq = np.zeros(len(fresp))
    integ_of_fresp_sq[0] = fresp[0]**2.0
    for i in range(1, len(fresp)):
        integ_of_fresp_sq[i] += integ_of_fresp_sq[i-1] + fresp[i-1] ** 2
    return integ_of_fresp_sq[len(integ_of_fresp_sq)-1]*bw
```

Verifying ADC filter Noise Bandwidth

The calculated noise bandwidth can be verified by applying broadband noise to the ADC input and measuring the total output noise. The figure beneath shows a 1000µV/√Hz band of noise applied to the AD7124 input, and the measured output data. The total output noise is 5.1mVRMS, close to the predicted value of 5.69mVRMS.

Conclusion

The techniques detailed in this paper are, individually, nothing new. But the simultaneous existence of:

- A large body of historical literature that over-emphasizes the importance of quantization noise and "getting all the bits you paid for"
- Modern, thermal noise limited ADCs that have "more than enough bits" to push quantization noise below thermal noise.
- Machine learning and artificial intelligence-based algorithms that allow circuit designers to under-emphasize sensor and signal chain performance

make it worthwhile to collect a few fundamental, easy to implement, and low-cost techniques to enable signal chain modeling and verification.

References

Smith, Steven W, *The Scientist & Engineer's Guide to Digital Signal Processing* <https://www.analog.com/en/education/education-library/scientist_engineers_guide.html>

Harris, Fredric, *On the use of windows for harmonic analysis with the discrete Fourier transform* Proceedings of the IEEE 66(1):51 - 83 <https://ieeexplore.ieee.org/document/1455106?arnumber=1455106>

Man, Ching, *Quantization Noise: An Expanded Derivation of the Equation, SNR = 6.02 N + 1.76* <https://www.analog.com/media/en/training-seminars/tutorials/MT-229.pdf>

Kester, Walt, *Taking the Mystery out of the Infamous Formula, "SNR = 6.02N + 1.76dB"* Analog Devices Tutorial, 2009. <https://www.analog.com/media/en/training-seminars/tutorials/MT-001.pdf>

Kester, Walt, *Oversampling Interpolating DACs* Analog Devices Tutorial, 2009. <https://www.analog.com/media/en/training-seminars/tutorials/MT-017. pdf>

Ruscak, Steve and Singer, L, *Using Histogram Techniques to Measure ADC Noise* Analog Dialogue, Volume 29, May, 1995.

<https://www.analog.com/en/analog-dialogue/articles/histogram-techniques- measure-adc-noise.html>

Active Learning Lab Activity: Analog to Digital Conversion <https://wiki.analog.com/university/courses/electronics/electronics-lab-adc>

Active Learning Tutorial: Converter Connectivity Tutorial <https://wiki.analog.com/university/labs/software/iio_intro_toolbox>

Acknowledgements

Jesper Steensgaard - enabled/forced a paradigm shift in thinking about signal chain design, starting with the LTC2378-20.

Travis Collins - Architect of Pyadi-iio (among many other things)

Adrian Suciu - Software Team Manager and contributor to libm2k

www.PosterPresentations.com