Mini Tutorial

ミックスドモード・シグナル・チェーンのノイズ解析にPythonを使用する簡単な方法

Mark Thoren、システム設計／アーキテクチャ・エンジニア
Cristina Șuteu、SWシステム設計エンジニア

概要

物理的世界の微妙な測定を伴うあらゆるアプリケーションは、正確で精度の高い低ノイズのシグナル・チェーンから始まります。近年の高集積データ・アクイジション・デバイスは、センサー出力に直接接続できるものが多く、アナログ・シグナル・コンディショニング、デジタル化、デジタル・フィルタリングを1つのシリコン・デバイス上で行って、システムの電子回路を大幅に単純化します。しかし、これらの最新デバイスでも、そこから最大限の性能を引き出したりデバッグを行ったりするには、やはりシグナル・チェーンのノイズ源とノイズ制限フィルタを完全に理解する必要があります。

はじめに

このチュートリアルは、コンバータ接続チュートリアル[1,2]の延長です。ここでは、個々のシグナル・チェーン要素のノイズに焦点を当て、Python/SciPy[3]とLTspice®を使ってそれらをモデル化します。更に、Pythonを使い、libm2kおよびLinux®産業用入出力（IIO）フレームワークを介してADALM2000多機能USBテスト装置を駆動することにより、結果を検証します。すべてのソース・コードと補足的な検討については、関連記事「アクティブ・ラーニング・ラボ演習」を参照してください。

ミックスドモード・シグナル・チェーンはあらゆる場所で使われています。簡単に言うと、現実の世界の信号を電気的表現に変換して、更にそれをデジタル化するシステムは、すべてミックスドモード・シグナル・チェーンに分類できます。信号は、このチェーンに沿ったあらゆるポイントにおいて様々な形で劣化しますが、通常この劣化は、何らかの形の歪みや追加ノイズによって特性評価することができます。デジタル領域に入ってしまうと、デジタル化されたデータの処理も決して完全なものではないとはいえ、少なくとも、実用的なあらゆる目的に関する限り、アナログ信号に有害な数多くの要素（部品公差、温度ドリフト、隣接信号からの干渉、電源電圧の変動など）に影響されるおそれはなくなります。

産業界が物理的な限界の拡大を続けるにあたって、確実なことが1つあります。計測器用のアナログおよびミックスド・シグナル部品には、常に改善の余地があるということです。スピード、ノイズ、消費電力、精度、あるいは価格の面で最先端技術を進歩させるようなA/Dコンバータ（ADC）やD/Aコンバータ（DAC）が市場に出現した場合、メーカーはそれを喜んで既存の問題に採用し、その後に更なる改善を求めるでしょう。しかし、アプリケーションに合った最良のアクイジション・システムを実現するには、コンポーネントの制限を知り、それに従ってコンポーネントを選ぶことが基本です。

一般的なミックスドモード・シグナル・チェーン

精密計測器アプリケーションの代表的な基本シグナル・チェーンを図1に示します。入力はアナログで、出力はデジタルです。ADCで使用できるバックグラウンド・リファレンスは多数あり[4]、ほとんどの読者が、ADCはある時点で入力信号をサンプリングして（あるいはある観測時間内における信号の平均を測定して）、信号の数値表現を作成すると理解しているでしょう。ほとんどの場合、この数値表現はゼロから$2^{(N-1)}$までの間の二進数として表現されます。ここで、Nは出力ワードのビット数です。

ADCのノイズ源

図1には複数のノイズ源がありますが、見過ごされたり過大に強調されたりしがちなのが、ADCのデジタル出力のビット数です。歴史的に見て、ADCのビット数は最終的な性能指数であると考えられており、16ビット・コンバータは14ビット・コンバータより4倍優れていると見なされていました[5]。しかし、現代の高分解能コンバータの場合は、ビット数を無視しても問題ありません。シグナル・チェーン設計の一般原則は次のようなものです。

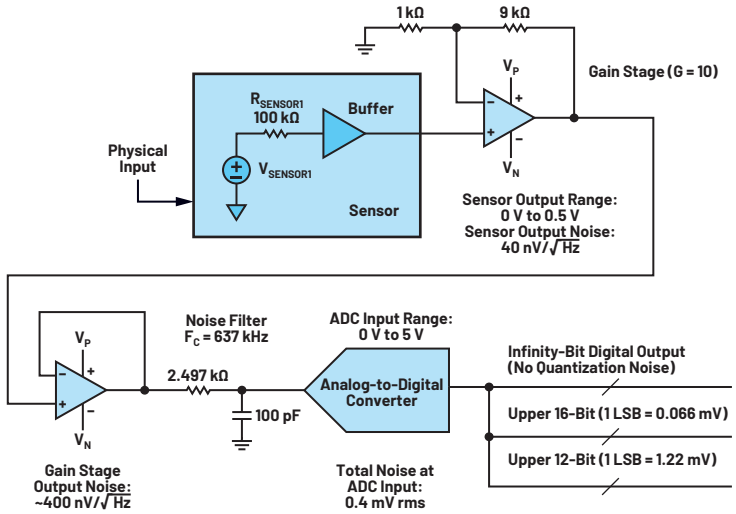

図1. ミックスドモード・シグナル・チェーンでは、温度、光の強度、pH、力、またはトルクといったいくつかの物理現象が、電気的パラメータに変換されます（抵抗または電流への変換、または電圧への直接変換）。この信号はその後増幅されてローパス・フィルタで処理され、更にADCによってデジタル化されます。デジタル化処理には、ADC内部でのデジタル・フィルタリングが含まれる場合があります。

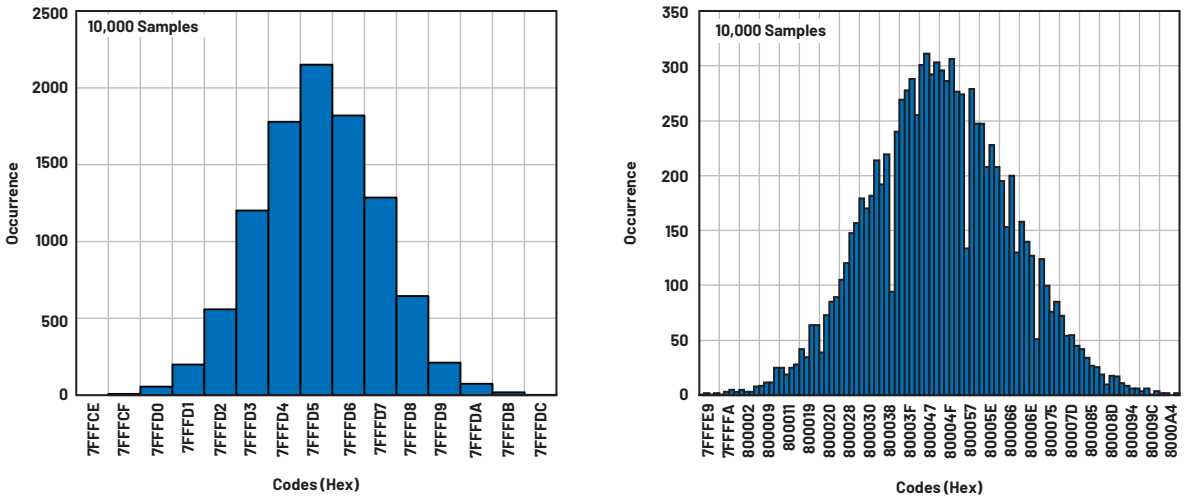

図2. PGAゲインが1の場合（左）はAD7124の出力ノイズは13個のコードで表され、標準偏差は約2.5コードです。量子化ノイズが見られますが、熱ノイズの方がより顕著です。PGAゲインが128の場合（右）は187個のコードで表され、量子化ノイズは大きくありません。また、1個または2個の最下位ビットを切り詰めても（量子化ノイズが2倍または4倍になる）、情報が失われることはありません。

「1つの段の入力ノイズは、その前段の出力ノイズよりある程度は小さくなければならない」

あらゆるシグナル・チェーンと同様に、多くの場合はADC内の1つのノイズ源が支配的になります。したがって、NビットADCにノイズのない信号が入力された場合は次のいずれかになります。

- 出力は1つのコードまたは2つの隣接するコードとなり、量子化ノイズが支配的となります。S/N比（SNR）が(6.02 N + 1.76)dBより大きくなることはありません[6]。
- 出力は多数のコードのガウス分布となり、熱ノイズ源が支配的となります。SNRは下の式で表される値以下になります。

$$20\ log\left(\frac{V_{IN}\ (p\text{-}p)}{\frac{\sigma}{\sqrt{8}}}\right) \quad (1)$$

ここで、

V_{IN} (p-p)はフルスケール入力信号、

σはボルト単位で表した出力コードの標準偏差です。

このすぐ後に例として使用するAD7124-8のように非常に高い分解能を持つコンバータが量子化ノイズによって制限されることは稀で、すべてのゲイン／帯域幅設定で熱ノイズが支配的になり、短絡入力では常に出力コードのきれいなガウス分布が生成されます。AD7124-8（24ビット・シグマ・デルタADC）の接地入力ヒストグラムを図2に示します。これらはそれぞれ内部プログラマブル・ゲイン・アンプ（PGA）を1と128に設定した時の図です。

ADCノイズのモデル化と測定

熱ノイズで制限されたADCのノイズは容易にモデル化できます。ノイズが「良好な挙動」（図2に示すようなガウス分布）を示すもので、なおかつADCの全入力範囲にわたり一定である場合、そのADCの時間領域ノイズは、図3に示すように、正規分布に従って乱数を生成するNumPy[7]の正規分布に従って乱数を生成する関数を使ってモデル化し、標準偏差を取ることによって検証することができます。

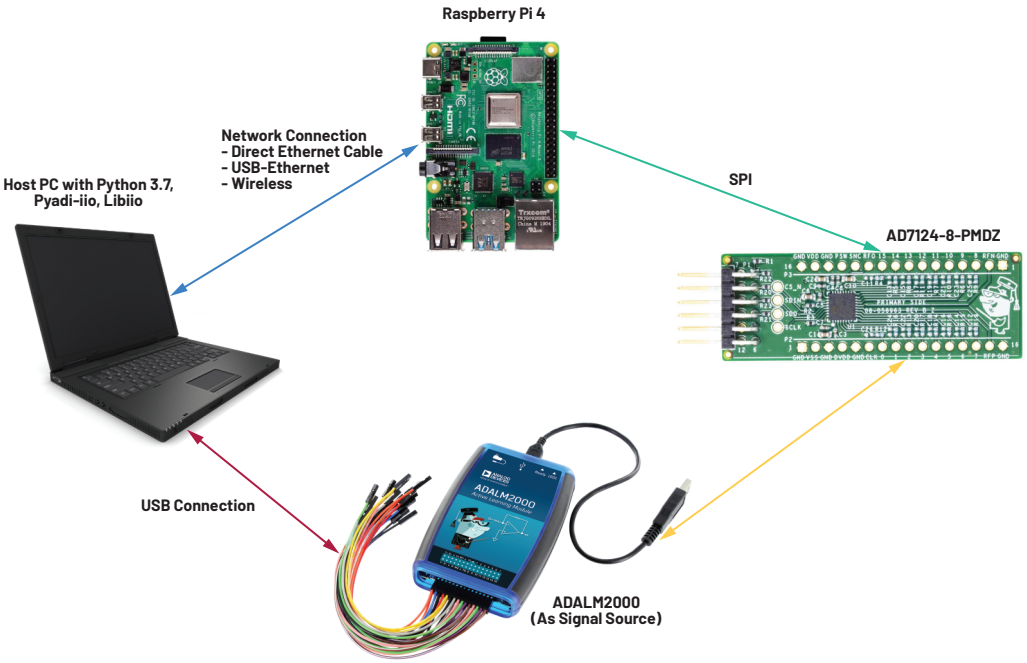

図4. ADALM2000は、2個の汎用アナログ入力と2個の出力を備えた多機能USBテスト装置で、サンプル・レートは入力が100MSPS、出力が150MSPSです。ADCのノイズ帯域幅とフィルタ応答を測定するためのシンプルな信号源として使用できます。AD7124デバイス・ドライバに対応したカーネルを実行するRaspberry Pi 4は、AD7124とホスト・コンピュータ間のシンプルなブリッジとして機能します。

```
# Model Gaussian Noise
# See AD7124 datasheet for noise levels per mode
import numpy as np
offset = 0.000
rmsnoise = 0.42e-6  # AD7124 noise
noise = np.random.normal(loc=offset, scale=rmsnoise,
                         size=1024)
measured_noise = np.std(noise)
print("Measured Noise: ", measured_noise)
```

図3　NumPyを使ったガウス・ノイズのモデル化。

AD7124のデバイス・ドライバは業界標準のIIOフレームワークに属するもので、安定したソフトウェアAPI（Pythonバインディングを含む）を備えています。アプリケーション・コードは、ローカルで（Raspberry Pi上で）実行するか、ネットワーク接続、シリアル接続、またはUSB接続を介してリモート・マシン上で実行することができます。更に、pyadi-iio[8]抽象化層がIIOデバイスとの連係に必要なボイラープレートのセットアップを処理して、ソフトウェア・インターフェースを大幅に簡略化します。AD7124-8への接続を開いてその設定を行い、データのブロックをキャプチャしてから接続を閉じる方法を、図5に示します。

```
 # AD7124-8 Basic Data Capture

import adi  # pyadi-iio library
# Connect to AD7124-8 via Raspberry Pi
my_ad7124 = adi.ad7124(uri="ip:analog.local")
ad_channel = 0  # Set channel
# Set PGA gain
my_ad7124.channel[ad_channel].scale = 0.0002983
my_ad7124.sample_rate = 128  # Set sample rate
# Read a single "raw" value
v0 = my_ad7124.channel[ad_channel].raw
# Buffered data capture
my_ad7124.rx_output_type = "SI"  # Report in volts
# Only one buffered channel supported for now
my_ad7124.rx_enabled_channels = [ad_channel]
my_ad7124.rx_buffer_size = 1024
my_ad7124._ctx.set_timeout(100000)  # Slow
data = my_ad7124.rx()  # Fetch buffer of samples

print("A single raw reading: ", v0)
print("A few buffered readings: ", data[:16])
del my_ad7124  # Clean up
```

図5　AD7124-8の基本データ・キャプチャ

AD7124-8への通信が確立された状態では、極めてシンプルですが極めて有効なテストを行うことができます。つまり、入力ノイズを直接測定します。これは、ADCへの入力を短絡して得られるADCコードの分布を見るだけですが、シグナル・チェーン設計の特性評価における重要なステップです。AD7124の入力モードはユニポーラに設定されているので、正の値だけが有効です。図6に示すテスト回路の入力は常に正になります。

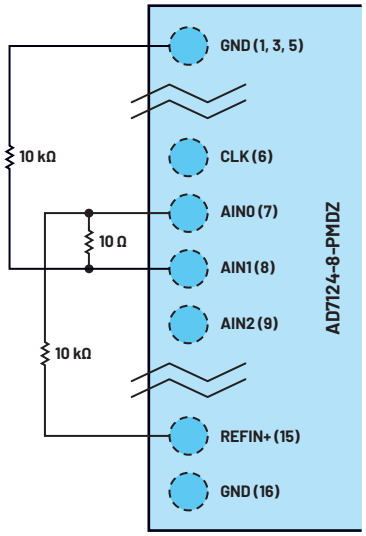

図6　抵抗分圧器を使ってAD7124-8の入力に1.25mVのバイアスをかけ、15µVの最大オフセット電圧を打ち消すことでADCにおける指示値が常に正になるようにします。

1024ポイントの測定結果を図7に示します。下側（青）のトレースは、最初に電力を供給した直後の測定結果です。

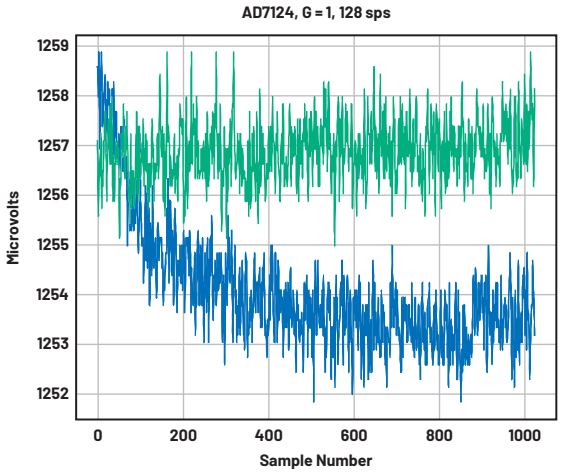

図7　AD7124-8のデータ・キャプチャは1.25mVのバイアスをかけて行いました。下側のトレースは、回路ウォームアップとしてパワーアップした後の初期ドリフトを示しています。上側のトレースは、30分間のウォームアップ終了後の安定した指示値です。

「変動」の原因としては、内部リファレンスのウォームアップ、外部抵抗のウォームアップ（およびそれによるドリフト）、あるいは寄生熱電対といった複数の要素が考えられます。寄生熱電対は、熱勾配が存在する状態で、わずかに異なる金属の間に電圧が生じる現象です。ウォームアップ後の測定ノイズは約565nV rmsで、これはデータシートのノイズ仕様と同等のレベルです。

ADCノイズを密度として表現

アナログ・シグナル・チェーン設計の一般原則（1つの段の入力ノイズはその前段の出力ノイズよりある程度は小さくなければならない）については、適切に仕様規定されたセンサーのほとんどと、ほぼすべてのアンプがそうであるように、すべての要素にノイズ密度仕様が含まれていれば、計算が容易です。

アンプやセンサーと異なり、通常、ADCのデータシートにはノイズ密度仕様が記載されていません。ADCのノイズを密度として表せば、アナログ・シグナル・チェーンの最終要素（ADCドライバ段、ゲイン段、センサー自体など）の出力のノイズと直接比較することができます。

ADCの内部ノイズは、DCからサンプル・レートの半分までの間のどこかに必ず現れます。理論的にはこのノイズはフラットです。あるいは、少なくとも形状を予測できます。実際には、ADCの合計ノイズは既知の帯域幅に拡散するので、シグナル・チェーン内の他の要素と直接比較可能なノイズ密度に変換できます。通常、高精度コンバータでは、合計ノイズがV rmsで直接規定されています。

$$e_{RMS} = \sigma \tag{2}$$

ここでe_{RMS}は合計rmsノイズで、コードの接地入力ヒストグラムの標準偏差から計算されます。

正弦信号によってテストと特性評価が行われる比較的高速のコンバータでは、通常、S/N比が仕様規定されています。この値が規定されている場合、合計rmsノイズは次式で計算できます。

$$e_{RMS} = \frac{ADCp\text{-}p}{\sqrt{8} \times 10^{\frac{SNR}{20}}} \tag{3}$$

ここで、ADCp-pはADCのピークtoピーク入力範囲です。

これから、次式で等価ノイズ密度を計算できます。

$$e_n = \frac{e_{RMS}}{\sqrt{\frac{f_S}{2}}} \tag{4}$$

ここで、f_SはADCのサンプル・レート（サンプル数／秒）です。

ウォームアップ後の図7の合計ノイズは、128SPSのデータ・レートで565nVでした。ノイズ密度はほぼ次のようになります。

$$\frac{565\ \text{nV}}{\sqrt{(64\ \text{Hz})}} = 70\ \frac{\text{nV}}{\sqrt{\text{Hz}}} \tag{5}$$

これでシグナル・チェーンのノイズ解析にADCを直接含めることができますが、これは、次に示すようなシグナル・チェーンのゲインを最適化するためのガイドラインとなります。

- ADC前段のノイズ密度がADCのノイズ密度よりわずかに高くなるポイントまでゲインを上げ、そこで止めます。シグナル・チェーンのゲインを、あえてこれより上げることはしないでください。ノイズを増幅して許容入力範囲を狭めることになります。

これは、ADCの入力範囲を「フルに使う」という通念には反しています。ADCの伝達関数に段差や不連続点が存在する場合は、より多くのADC入力範囲を使用する方が有利ですが、「良好な挙動」のADC（ほとんどのシグマ・デルタADCと最新の高分解能逐次比較レジスタ（SAR）ADC）に望ましいアプローチは、ノイズによる最適化です。

ADCフィルタ応答の測定

AD7124-8はシグマ・デルタADCで、その内部では変調器が高サンプリング・レートでアナログ入力表現を生成しますが、ノイズが多く（分解能が低く）なります。このノイズの多いデータが内部デジタル・フィルタによって処理されて、低レート低ノイズの出力が生成されます。フィルタのタイプは、意図するエンド・アプリケーションによってADCごとに異なります。AD7124-8は汎用デバイスですが、高精度アプリケーションをターゲットにしています。したがって、デジタル・フィルタ応答と出力データ・レートは自由度の高い設定が可能です。フィルタ応答はデータシートに詳しく規定されていますが、フィルタが所定の信号に与える影響を測定したい、という場合も考えられます。AD7124-8のフィルタ応答コード・ブロック（図9を参照）は、ADC入力にサイン波を加え、その出力を解析することによってフィルタ応答を測定します。この方法は、他の波形（ウェーブレットやシミュレートされた物理的事象）の測定に簡単に応用できます。ADALM2000は、図8に示す要領でAD7124-8回路に接続します。

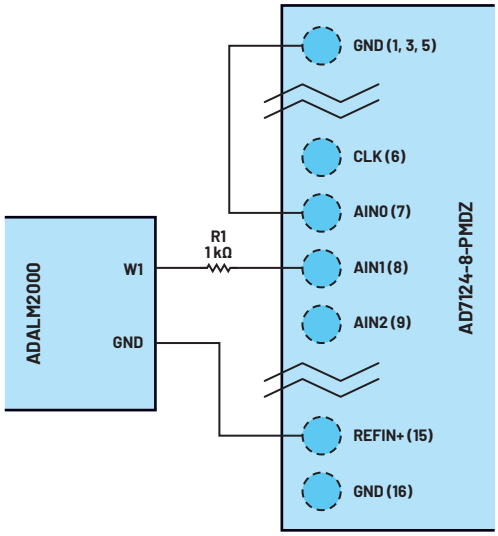

図8　ADALM2000波形発生器は一定範囲のサイン波周波数を生成するために使われ、AD7124-8のフィルタ応答を直接測定できます。スクリプトがサイン波の振幅とオフセットを安全なレベルに設定し、1kΩの抵抗は不具合発生時にAD7124-8を保護します。（ADALM2000の出力電圧範囲は–5V～+5Vで、AD7124-8の絶対最大制限値は–0.3Vと+3.6Vです）

AD7124-8のフィルタ応答コード・ブロック（図9参照）は、ADALM2000の波形発生器が10Hzのサイン波を生成し、1024個の応答データ・ポイントをキャプチャして、rms値を計算し、更にその結果をリストに追加するように設定します。send_sinewaveとcapture_dataはユーティリティ関数で、前者はADALM2000にサイン波を送り、後者はAD7124からデータ・ブロックを受け取ります[2]。更に周波数を変えながら120Hzまで同じことを繰り返して、図10に示すようにその結果をプロットします。

```
# AD7124-8 Filter Response
import numpy as np
import matplotlib.pyplot as plt
resp = []
freqs = np.linspace(1, 121, 100, endpoint=True)
for freq in freqs:
    print("testing ", freq, " Hz")
    send_sinewave(my_siggen, freq)   # Set frequency
    time.sleep(5.0)                  # Let settle
    data = capture_data(my_ad7124)   # Grab data
    resp.append(np.std(data))   # Take RMS value
    if plt_time_domain:
        plt.plot(data)
        plt.show()
    capture_data(my_ad7124)   # Flush
# Plot log magnitude of response.
response_dB = 20.0 * np.log10(resp/0.5*np.sqrt(2))
print("\n Response [dB] \n")
print(response_dB)
plt.figure(2)
plt.plot(freqs, response_dB)
plt.title('AD7124 filter response')
plt.ylabel('attenuation')
plt.xlabel("frequency")
plt.show()
```

図9　ADALM2000のフィルタ応答ブロック・プログラム。

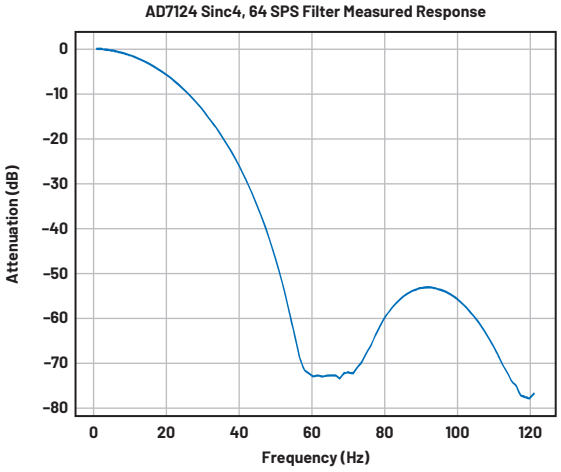

図10　64SPS、sinc4モードでのAD7124フィルタ応答の測定結果には、フィルタのパス・バンド、第1ローブ、および最初の2つのヌルが示されています。

高減衰値を測定するには低ノイズの歪みの小さい信号発生器が必要ですが、このセットアップでも最初のいくつかの主要ローブの応答は明らかです。

ADCフィルタのモデル化

ADCのフィルタ機能を測定できる機能は、ベンチ検証のための実用的なツールです。しかし、シグナル・チェーン全体をシミュレートするには、フィルタのモデルが必要です。多くのコンバータ（AD7124-8を含む）では、これが明示的に提供されていませんが、データシートに記載された情報からリバース・エンジニアリングすることによって、使用可能なモデルを作成できます。

以下に示すのはAD7124-8のモデルに過ぎず、ビットを正確に再現したものではありません。すべての規定パラメータについては、AD7124-8のデータシートを参照してください。

AD7124のすべてのフィルタは、様々なsinc機能を組み合わせた周波数応答性を備えています（(sin{f}/f)^{N}に比例した周波数応答）。これらのフィルタは構成が極めて容易で、ヌルが分かっていればリバース・エンジニアリングも容易です。

AD7124-8の10Hzノッチ・フィルタを図11に示します。より高次のsinc3およびsinc4フィルタを組み合わせたものも使用できます。

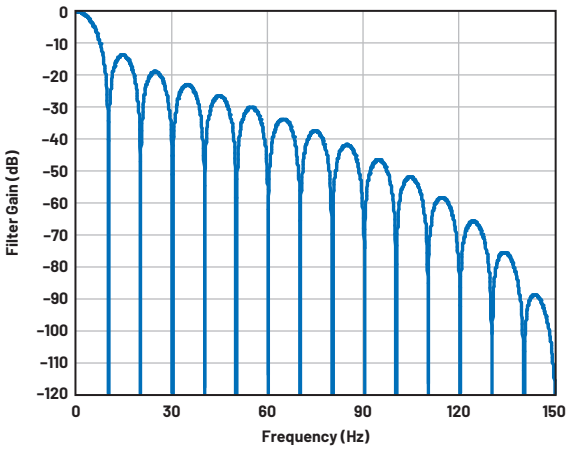

図11　AD7124-8の10Hzノッチ・フィルタはsinc1振幅応答です。このフィルタのインパルス応答は、時間100msで取得したサンプルの単純な非加重（矩形）平均です。

図12に示す50Hz/60Hz同時除去フィルタは非自明な例です。このフィルタは、50Hz（ヨーロッパなど）または60Hz（米国など）のAC電力ラインからノイズを強力に除去することを目的としたものです。

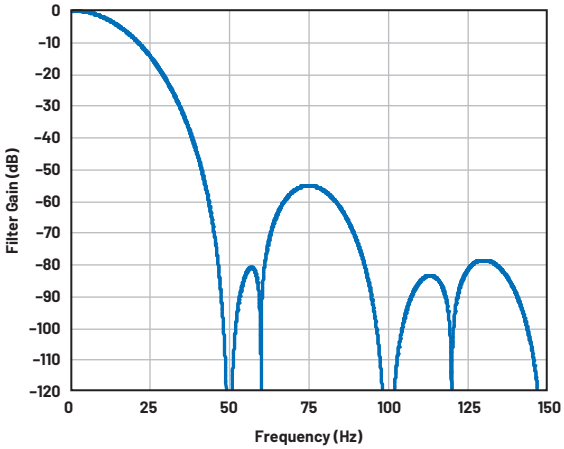

図12　AD7124-8の50Hz/60Hz除去フィルタの応答は、50Hz sinc3フィルタと60Hz sinc1フィルタの組み合わせです。

より高次のsincフィルタは、sinc1フィルタを畳み込むことによって作成できます。例えば、2つのsinc1フィルタ（時間矩形波インパルス応答のもの）を畳み込むと三角波インパルス応答となり、これに応じたsinc2周波数応答が得られます。AD7124のフィルタ・コード・ブロック（図13を参照）は、50Hzにヌルのあるsinc3フィルタを生成してから、60Hzにヌルのある4番目のフィルタを追加します。

```
# AD7124 Filters
import numpy as np
f0 = 19200
# Calculate SINC1 oversample ratios for 50, 60Hz
osr50 = int(f0/50)   # 384
osr60 = int(f0/60)   # 320

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

図13　AD7124-8の50Hz/60Hz sincフィルタのコード例。

この場合に得られるインパルス（時間領域）の形状を図14に示します。フィルタ係数（タップ）の値は、ゼロ周波数でユニティ（0dB）ゲインに正規化されます。

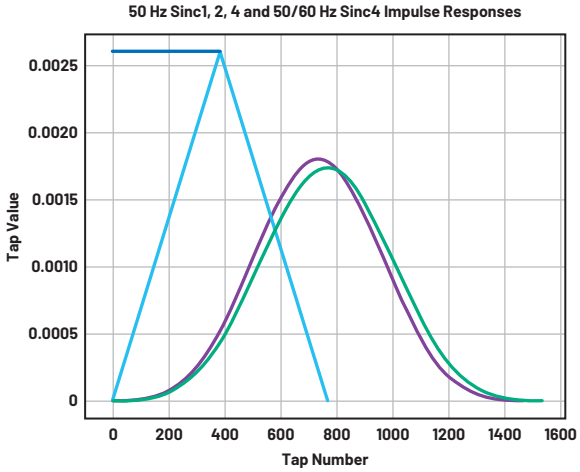

図14　矩形波インパルス応答を繰り返し畳み込むと、まず三角波応答となり、更にガウス状のインパルス応答となります。

最終的には、図16に示すようにNumPyのfreqz関数を使って周波数応答を計算できます。図15に応答を示します。

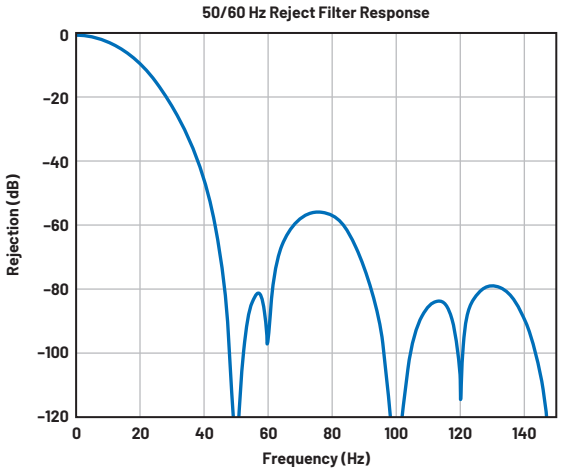

図15　sinc1 60Hzフィルタでsinc3 50Hzノッチ・フィルタを畳み込むと、50Hzと60Hzの両方を強力に除去する複合応答が得られます。

```
# AD7124 Frequency Response
import numpy as np
from scipy import signal
f0 = 19200
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)
```

図16　AD7124-8でsinc3 50Hzノッチ・フィルタとsinc 60Hzフィルタを組み合わせるコードの例

抵抗は無駄：センサーの基本的な限界

すべてのセンサーには、その完全性に関わりなく、何らかの最大入力値（およびそれに対応する、電圧、電流、抵抗、あるいは場合によってはダイヤル位置などの最大出力）と有限ノイズ・フロア（入力にまったく変動がない場合でも存在する出力の微細な変動）があります。電気的出力を持つセンサーには、図17のR_{SENSOR}で表される有限抵抗（より一般的にはインピーダンス）を持つ要素がどこかに含まれます。これは改善し得ない1つの基本

的なノイズ限界を表しており、この抵抗は少なくともen(RMS)ボルトのノイズを発生させます。

$$e_n(RMS)=\sqrt{4\times K\times T\times R_{SENSOR}\times(F2-F1)} \quad (6)$$

ここで、

e_N(RMS)は合計ノイズ、

kはボルツマン定数（1.38^{e-23}J/K）、

Tは抵抗の絶対温度（ケルビン）、

F2とF1は対象周波数帯の上限値と下限値で、

帯域幅を1Hzに正規化するとノイズ密度が得られます（V/√Hz）。

センサーのデータシートには低出力インピーダンス（多くの場合は0Ωに近い）が仕様規定されていることがありますが、多くの場合これはバッファ段です。これにより下流側回路とのインターフェーシングは容易になりますが、シグナル・チェーン初期段階のインピーダンスによるノイズは除去されません。

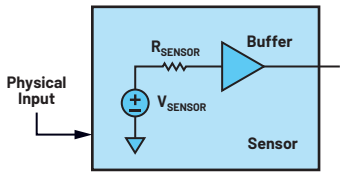

図17　多くのセンサーは、下流側回路との接続を容易にするためにバッファを内蔵しています。出力インピーダンスは小さくなりますが（多くの場合は0Ωに近付く）、高インピーダンスの検出素子によるノイズも信号と共にバッファされます。

センサーには他にも様々な制約があり（機械的、化学的、光学的なもの）、それぞれが固有の理論的制限値を持っています。これらの影響はモデル化して、後で補償することができます。しかし、ノイズは補償することのできない不完全性の1つです。

ラボのノイズ源

補正済みのノイズ発生器は、実際には何も検知しないセンサーのノイズをエミュレートする、「世界最悪のセンサー」として機能します。このような発生器があれば、ノイズに対するシグナル・チェーンの応答を直接測定することができます。図18に示す回路は、「問題のない」精度と帯域幅を持つ127nV/√Hz（室温時）のノイズ源として、1MΩの抵抗を使用しています。精度は「問題のない」範囲に止まりますが、この方法には次のような利点があります。

- 第1の原則に基づいているので、ある意味では補正されていない標準の役割を果たします。
- これは本当の意味でランダムであり、繰り返しパターンがありません。

OP482は超低バイアス電流のアンプです。バイアス電流が小さいので電流ノイズも小さく、更に電圧ノイズも十分に小さいので、1MΩの入力インピーダンスによるノイズが支配的になります。2121のゲインで構成した場合、出力ノイズは269μV/√Hzです。

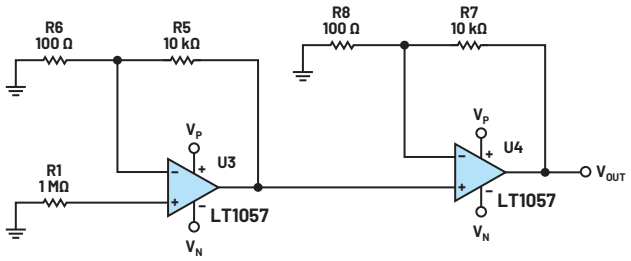

図18　1MΩの抵抗は予測可能なノイズ源としての役割を果たし、これは低ノイズのオペアンプによって使用可能なレベルまで増幅されます。

ノイズ源の検証は、Scopy GUIのスペクトラム・アナライザを使い、ADALM2000 USB計測器で行いました。結果を図19に示します[9]。

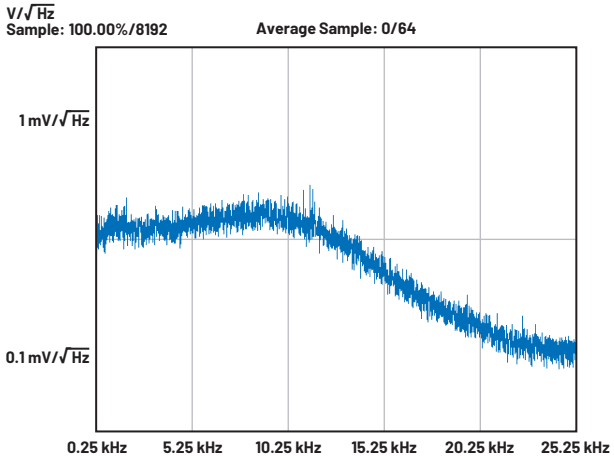

図19　抵抗ベースのラボ用ノイズ発生器の使用可能出力帯域幅は約10kHzです。

ここに示すアナライザ設定では、ADALM2000のノイズ・フロアは40μV/√Hzで、ノイズ源の269μV/√Hzより十分に低い値です。

Scopyは単一の目視測定に有効ですが、その機能はSciPyのピリオドグラム機能を使って簡単に複製できます。未加工データはlibm2k[10]とPythonバインディングを使ってADALM2000から収集され、最小限の処理を行ってDC成分を除去してから（除去しないと低周波数ビンに漏れ出します）、nV/√Hz単位にスケーリングされます。図20に示すこの方法は、サンプル・レートが固定された既知の値で、なおかつデータを電圧のベクトルとしてフォーマットできる限り、任意のデータ・アクイジション・モジュールに適用できます。

```
# Noise Source Measurement
import numpy as np
navgs = 32  # Avg. 32 runs to smooth out data
ns = 2**16
vsd = np.zeros(ns//2+1)  # /2 for onesided
for i in range(navgs):
  ch1 = np.asarray(data[0])  # Extract ch 1 data
  ch1 -= np.average(ch1)  # Remove DC
  fs, psd = periodogram(ch1, 1000000,
                        window="blackman",
                        return_onesided=True)
  vsd += np.sqrt(psd)
vsd /= navgs
```

図20　ADALM2000用のPythonノイズ源測定コード。

以上で、既知のノイズ源と、そのノイズ源を測定する方法の準備が整いました。これらは、共にシグナル・チェーンの評価に使用できます。

LTspiceでシグナル・チェーンをモデル化

LTspice®は無料で入手できる汎用アナログ回路シミュレータで、シグナル・チェーンの設計に利用できます。このツールはトランジェント解析、周波数領域解析（ACスイープ）、ノイズ解析を行うことができ、その結果をエクスポートして、Pythonを使いミックスド・シグナル・モデルに組み込むことができます。

図21はアナログ・ノイズ発生器のノイズ・シミュレーションの結果で、実験結果とほぼ一致しています。このシミュレーションには、OP482と同様の特性を持つオペアンプを使用しました。

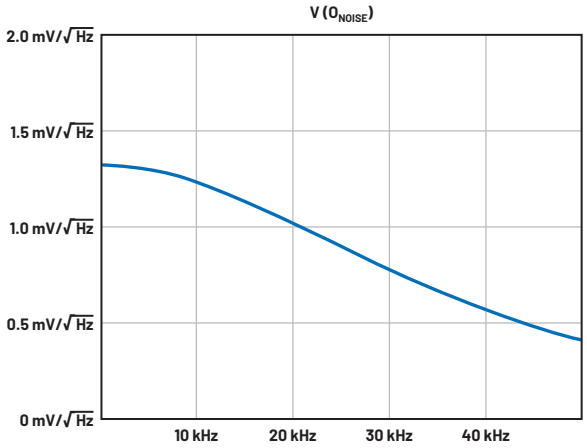

図21　ラボのノイズ源のLTspiceシミュレーションは、測定回路とほぼ同じ使用可能帯域幅を示しています。

図22の回路ノイズは、ある周波数帯（対象の信号が存在する帯域）では一定で、それより高い帯域ではほぼ1次ローパス応答でロールオフするという点を考えると、モデル化する上で極めて一般的なものと言えます。この手法が役立つのは、高次のアナログ・フィルタリングやアクティブ素子自体が原因でフラットではなくなったノイズ・フロアをモデル化する場合です。その典型的な例が、LTC2057などのオートゼロ・アンプによく見られるノイズの山です。図23を参照してください。

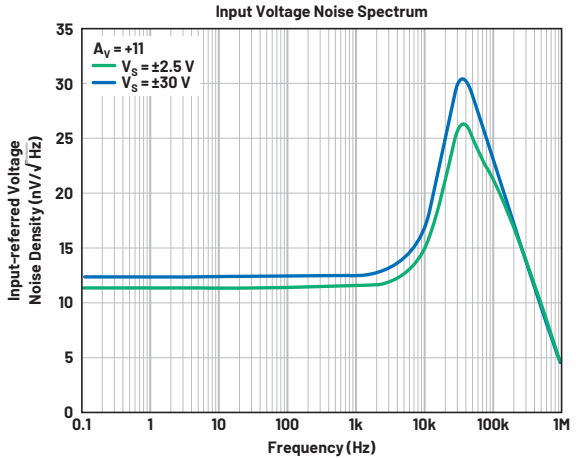

図22　LTC2057のノイズ密度は低周波数域ではフラットで、50Hz（内部発振器の周波数100kHzの半分）にピークがあります。

Pythonで周波数領域解析を行うためにLTspiceのノイズ・データをインポートする場合、重要なのは、分析対象の周波数帯を正確にシミュレートできるようにシミュレーション・コマンドをセットアップすることです。この場合、ノイズ・シミュレーションは、最大周波数2.048MHz、分解能62.5Hzでセットアップします。これは、サンプル・レート4.096MSPSでの第1ナイキスト・ゾーンに相当します。非反転ゲイン10、シミュレーション出力、エクスポート・データ・フォーマットでLTC2057をシミュレートした結果を図23に示します。

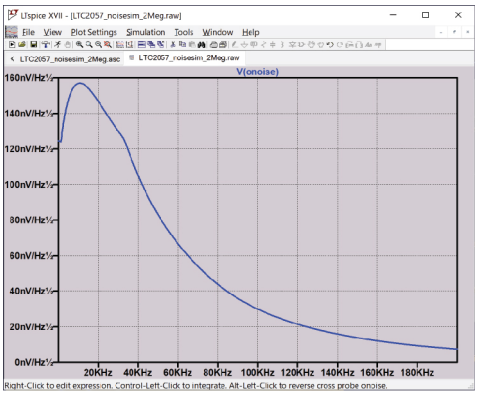

図23　非反転ゲインを+10に設定したLTC2057の出力ノイズをLTspiceでシミュレート。LTspiceはノイズを積分するためのシンプルなツールを提供しますが、シミュレーションの結果は、更に解析を行うためにエクスポートして、Pythonにインポートすることができます。

所定帯域のノイズが信号に与える影響（S/N比）を決定するために、ノイズは対象帯域幅内で二乗和平方根で積分されます。LTspiceでは、プロット限界を設定して、コントロール・キーを押しながらパラメータ・ラベルをクリックすることにより、プロットしたパラメータを積分できます。2.048MHzシミュレーション全体での合計ノイズは32µV rmsです。この動作をPythonで行う関数を図24に示します。

```
def integrate_psd(psd, bw):
    import numpy as np
    int_psd_sqd = np.zeros(len(psd))
    integrated_psd = np.zeros(len(psd))
    int_psd_sqd[0] = psd[0]**2.0
    for i in range(1, len(psd)):
        int_psd_sqd[i] += int_psd_sqd[i-1]\
            + psd[i-1] ** 2
        integrated_psd[i] += int_psd_sqd[i]**0.5
    integrated_psd *= bw**0.5
    return integrated_psd
```

図24　二乗和平方根を実装するPythonコード

エクスポートされたノイズ・データを読み込んでintegrate_psd関数に渡すと合計ノイズは3.21951e-05となり、LTspiceの計算に非常に近い結果が得られます。

テスト・ノイズの生成

純粋にアナログのノイズ発生器の機能を拡張する場合、フラットなノイズ・プロファイルだけでなく、任意のノイズ・プロファイルも生成できるようにすると非常に便利です。例えば、フラット・バンドのノイズ、ピンク・ノイズ、一部のアンプにおけるピークをエミュレートするノイズの山などです。図25のハーフスペクトラム・コード・ブロックの生成時系列は、必要ノイズ・スペクトラム密度（手動で作成するか、LTspiceシミュレーションから取り込むことができます）と時系列のサンプル・レートから始まり、次にDACに送信可能な電圧値の時系列を作成します。

```
def time_points_from_freq(freq, fs=1, density=False):
    import numpy as np
    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_s_full = np.concatenate((freq, np.roll
                               (np.flip(freq), 1)))
    r_spectrum_rnd_ph = r_s_full * rnd_ph_full
    r_time_full = np.fft.ifft(r_spectrum_rnd_ph)

    if (density is True):
        # Note that this N is "predivided" by 2
        r_time_full *= N*np.sqrt(fs/(N))
    return(np.real(r_time_full))
```

図25　任意ノイズ・プロファイルを作成するPythonコード。

この関数は、libm2kスクリプトを通じて1つのADALM2000を制御し、次いでもう1つのADALM2000とScopy GUIのスペクトラム・アナライザでノイズ・プロファイルを確認することによって検証できます。ADALM2000へノイズ時系列をプッシュするコード・スニペット（「Push Noise Time-series to ADALM2000」– 図26参照）は、（ダブルチェック機能用にW1にサイン波を加えた状態で）ADALM2000のW2出力上に1mV/√Hzノイズのバンドを4つ生成します。

```
# Push Noise Time-series to ADALM2000
import numpy as np
n = 8192
# create some "bands" of  1mV/rootHz noise
bands = np.concatenate((np.ones(n//16),
                        np.zeros(n//16),
                        np.ones(n//16),
                        np.zeros(n//16),
                        np.ones(n//16),
                        np.zeros(n//16),
                        np.ones(n//16),
                        np.zeros(n//16)))*1000e-6
bands[0] = 0.0  # Set DC content to zero
buffer2 = time_points_from_freq(bands, fs=75000,
                                density=True)
buffer = [buffer1, buffer2]
aout.setCyclic(True)
aout.push(buffer)
```

図26　ADALM2000で任意ノイズを検証。

1つのADALM2000によって生成される1mV/√Hzノイズの4つのバンドを図27に示します。入力ベクトルの長さは、ポイントあたり9.1Hzの帯域幅に対し、75kSPSのサンプル・レートで8192ポイントです。各バンドの幅は512ポイント（4687Hz）です。約20kHzを超えてからのロールオフはDACのsincロールオフです。DACがより高いサンプリング・レートに対応可能であれば、時系列データをアップサンプリングして、インターポレーション・フィルタによりフィルタ処理をすることができます[11]。

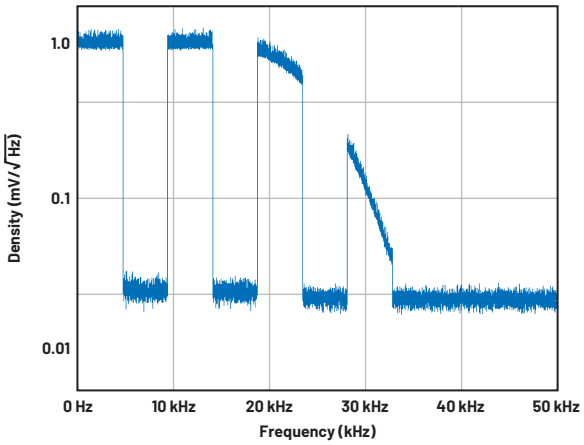

図27　Scopyのスペクトラム・アナライザは、任意ノイズ発生器を検証するために使用します。ノイズ・バンドの間にある深いノッチにはそのアナライザのノイズ・フロアが現れますが、これは任意ノイズ・プロファイルを正確に生成できることを示しています。

このノイズ発生器は、シグナル・チェーンの除去特性を検証するために、純粋なアナログ発生器と組み合わせて使用することができます。

ADCノイズ帯域幅のモデル化と検証

$f_S/2$を超える外部ノイズ源とスプリアス・トーンは、DCから$f_S/2$までの領域にフォールド・バック（エイリアス）し、コンバータが$f_S/2$よりはるかに高いノイズに敏感になることがあります。LTC2378-20の場合を考えてみます。このデバイスのサンプル・レートは1MSPSで、–3dB入力帯域幅は34MHzです。このように高い周波数では最良の性能を得られないことがありますが、このコンバータは68以上のノイズのナイキスト・ゾーンをデジタル化して、使用する信号の上にそれをフォールド・バックします。これは、広帯域ADC用のアンチエイリアシング・フィルタの重要性を示しています。精密アプリケーション用のコンバータは、通常、シグマ・デルタ・アーキテクチャ（AD7124-8など）かオーバーサンプリングSARアーキテクチャであり、その入力帯域幅は設計によって制限されます。

多くの場合、ADCの組み込みフィルタを含めて、フィルタの等価ノイズ帯域（ENBW）について考えることは有益です。ENBWは、フラットなパス・バンドを備えた「ブリック・ウォール」フィルタのうち、フラットではないフィルタと同量のノイズを通過させるものの帯域幅です。一般的な例が1次RCフィルタのENBWで、これは次式で表されます。

$$ENBW = \frac{f_C \times \pi}{2} \quad (7)$$

ここで、f_Cはフィルタのカットオフ周波数です。「DCから日光まで」の広帯域ノイズを1kHzの1次ローパス・フィルタと1.57kHzのブリック・ウォール・ローパス・フィルタ両方の入力に加えると、出力の合計ノイズ電力は同じになります。

図28のENBWサンプル・コード・ブロックは、フィルタの振幅応答を受け取って有効ノイズ帯域幅を返します。ENBW = $f_C \times \pi/2$の関係を検証するために、単極フィルタの振幅応答を計算して使用します。

```
import numpy as np
def arb_enbw(fresp, bw):
    int_frsp_sqd = np.zeros(len(fresp))
    int_frsp_sqd[0] = fresp[0]**2.0
    for i in range(1, len(fresp)):
        int_frsp_sqd[i] += (int_frsp_sqd[i-1] +
                            fresp[i-1] ** 2)
    return int_frsp_sqd[len(int_frsp_sqd)-1]*bw

fc = 1  # Hz
bw_per_point = 200/65536  # Hz/record length
frst_ord = np.ndarray(65536, dtype=float)
# Magnitude = 1/SQRT(1 + (f/fc)^2))
for i in range(65536):
    frst_ord[i] = (1.0 /
                   (1.0 +
                    (i*bw_per_point)**2.0)**0.5)
fo_enbw = arb_enbw(frst_ord, bw_per_point)
```

図28　有効ノイズ帯域幅を計算するPythonコードの例。

この関数は、AD7124の内部フィルタを含め、任意のフィルタ応答のENBWを計算するために使用できます。AD7124のsinc4フィルタ（128SPSサンプル・レート）の周波数応答は、前に述べた50Hz／60Hz除去フィルタの例と同様の方法で計算できます。arb_anbw関数は約31HzのENBWを返します。

この結果の検証にはADALM2000ノイズ発生器を使用できます。1000µV/√Hzのバンドを生成するようにテスト用ノイズ発生器を設定すると、合計ノイズは約5.69mV rmsとなるはずで、測定による合計ノイズは約5.1mV rmsです。ADC入力信号のオシロスコープ・キャプチャを、図29のADC出力データの隣に示します。測定されたピークtoピーク・ノイズは426mVで、ADCのピークtoピーク・ノイズは約26mVです。このような高レベルのノイズは実際の高精度シグナル・チェーンでは非現実的なもの（と願いたい）ですが、この演習は、ADCの内部フィルタはシグナル・チェーン内において帯域幅を制限し、それによりノイズを低減する要素としての役割を果たすと期待できることを示しています。

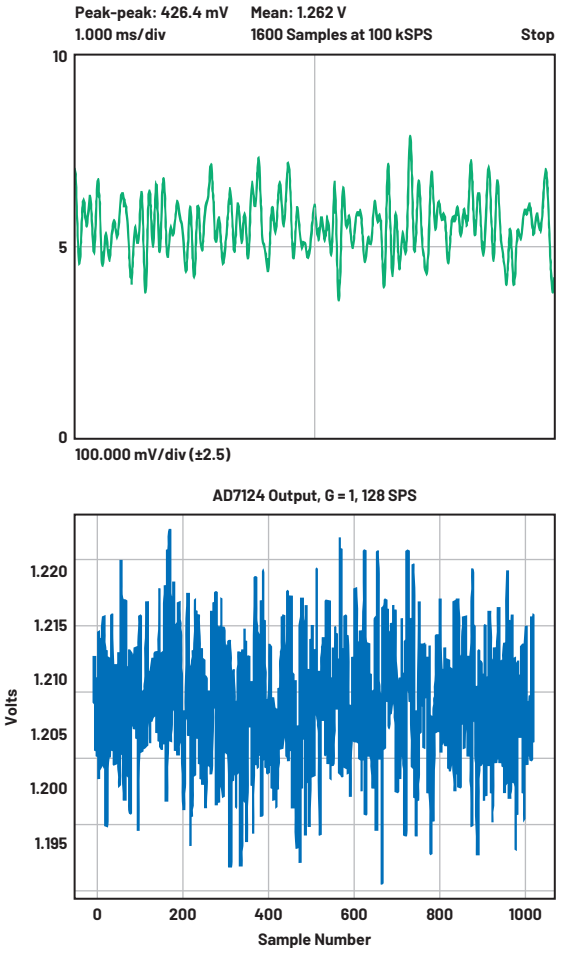

図29　1mV/√HzのノイズをAD7124-8に入力。ノイズの定性的減少は明らかで、ADC入力における426mVのピークtoピーク・ノイズは、ADC出力で約25mVになっています。ノイズ密度1mV/√Hzで、ADCのフィルタのENBWが31Hzの場合、5.1mV rmsの合計出力ノイズは予想値の5.69mV rmsに近い値です。

まとめ

ノイズはあらゆるシグナル・チェーンを制限する要素であり、ノイズによって信号が損なわれると情報が失われてしまいます。信号アクイジション・システムを構築する場合は、事前にアプリケーションの条件を理解し、それに応じてコンポーネントを選択して、プロトタイプ回路でテストをする必要があります。このチュートリアルでは、センサーとシグナル・チェーンのノイズを正確にモデル化して測定する一連の方法を示しました。これらの方法は、設計およびテストのプロセスに使用できます。

このチュートリアルに示した個々のテクニックは、決して新しいものではありません。しかし、適切なシステムを実現するためには、基本的で実現が容易な、しかもコストのかからない様々なテクニックを利用して、シグナル・チェーンのモデル化と検証をできるようにすることが重要です。各メーカーは性能が向上したパーツを提供し続けていますが、留意しなければならない一定の限界は常に存在します。ここに示したテクニックは、ミックスドモード・シグナル・チェーン構築前のパーツの妥当性確認に使用できるだけでなく、既存チェーンの設計上の欠陥を明らかにするために使用することもできます。

謝辞

- Jesper Steensgaard – LTC2378-20から始めて、シグナル・チェーン設計を考える際のパラダイム・シフトを可能にし、実現しました。
- Travis Collins – Pyadi-iioのアーキテクト（他にも多数の優れた実績があります）。
- Adrian Suciu – ソフトウェア・チーム・マネージャ兼libm2kへの貢献者。

参考資料

[1] "Converter Connectivity Tutorial." Analog Devices Wiki, January 2021.

[2] Analog Devices Education Tools Repository. Zenodo, July 2021.

[3] Pauli Virtanen, Ralf Gommers, et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python." Nature Methods, 17(3), February 2020.

[4] Steven W. Smith. The Scientist & Engineer's Guide to Digital Signal Processing. California Technical Publishing, 1999.

[5] Ching Man. "MT-229: Quantization Noise: An Expanded Derivation of the Equation, SNR = 6.02 N + 1.76." Analog Devices, Inc., August 2012.

[6] Walt Kester. MT-001：「S/N比＝6.02N + 1.76DB」、その意味と利用上の注意点。アナログ・デバイセズ、2009年

[7] Charles R. Harris, K. Jarrod Millman, et al. "Array Programming with NumPy." Nature, 585, September 2020.

[8] "pyadi-iio: Device Specific Python Interfaces for IIO Drivers." Analog Devices Wiki, May 2021.

[9] "Scopy." Analog Devices Wiki, February 2021.

[10] "What Is Libm2k?" Analog Devices Wiki, October 2021.

[11] Walt Kester. "MT-017: Oversampling Interpolating DACs." Analog Devices, Inc., 2009.

出典

本稿の内容は、2021年に開催された「Scientific Computing with Python」カンファレンスのプロシーディングに、「Using Python for Analysis and Verification of Mixed-Mode Signal Chains」という表題で初めて掲載されたものです。DOI: 10.25080/majora-1b6fd038-001.

著者について

Mark Thoren

高精度データ・コンバータを担当するアプリケーション・エンジニアとして、2001年にリニアテクノロジー（現アナログ・デバイセズの1部門）に入社しました。以降、評価システムの開発、トレーニング、技術文書の発行、カスタマ・サポートを含むミックスド・シグナル・アプリケーション分野で、様々な役割を果たしてきました。現在はアナログ・デバイセズのシステム開発グループに所属するシステム・エンジニアで、リファレンス設計とアナログ・デバイセズ・ユニバーシティ・プログラムに関する業務に就いています。オロノにあるメイン大学で、農業機械工学の学士号と電気工学の修士号を取得しました。

連絡先：mark.thoren@analog.com

Cristina Ștuteu

2019年にシステム・アプリケーション・エンジニアとしてアナログ・デバイセズに入社し、システム開発グループに配属されました。アナログ・デバイセズではソフトウェアの改善に貢献してきた他、技術文書の発行とトレーニングにおいて様々な役割を果たし、ADALM2000関連のビデオ・シリーズを含めて、アナログ・デバイセズ・ユニバーシティ・プログラム用の教材を提供してきました。ルーマニアのクルージュ・ナポカ工科大学で電気工学の学士号を、同大学とフランスのボルドー大学の共同研究において信号および画像処理の修士号を取得しています。

連絡先：cristina.suteu@analog.com

＊英語版技術記事はこちらよりご覧いただけます。