一种使用Python来分析混合模式信号链中的噪声的简单方法

Mark Thoren，系统设计/架构工程师
Cristina Șuteu，SW系统设计工程师

摘要

涉及对真实世界进行敏感测量的应用都是从准确、精密的低噪声信号链开始。现代高度集成的数据采集器件通常可以直接连接到传感器输出，在单个硅器件上执行模拟信号调理、数字化和数字滤波，这极大地简化了系统电子组成。但是，要使这些现代器件发挥出色性能，并对它们进行调试，我们仍然需要深入了解信号链的噪声源和噪声限制滤波器。

简介

本教程是转换器连接教程的续篇。[1,2] 将侧重介绍单个信号链元件的噪声，并使用Python/SciPy[3]和LTspice®来模拟这些噪声。然后，使用Python，通过libm2k和Linux®工业输入输出(IIO)框架来驱动ADALM2000多功能测试仪器来验证模拟结果。关于源代码和更多讨论，请参见配套的主动学习实验室练习。

混合模式信号链无处不在。简单地说，任何将真实世界的信号转换为电子表示（然后数字化）的系统都可以被归类为混合模式信号链。在信号链的每个点上，信号都以各种方式降级，从特征来看，可能是出现一定程度的失真，或是出现相加噪声。在进入数字领域之后，对数字化数据的处理也不是完美的，但至少，实际上可以不受许多影响模拟信号的因素的影响——部件公差、温度漂移、邻近信号的干扰或电源电压变化。

随着行业不断扩展物理限制，有一点是肯定的：仪器仪表的模拟和混合信号部件始终存在可改进的空间。如果市场上出现的模数转换器(ADC)或数模转换器(DAC)在速度、噪音、功率、精度或价格方面都表现出色，制造商会很乐意用其来解决现有问题，然后要求进行更多改进。但是，为了给您的应用提供最佳的采集系统，我们必须了解每种部件的限制，然后做出相应选择。

一种通用的混合模式信号链

图1显示了在精密仪器应用中很典型的一种通用信号链，提供物理输入和数字输出。目前有许多关于ADC的背景参考资料[4]，大部分读者都知道，ADC会在某个时点对输入信号进行采样（或测量某个观察时间段内信号的平均值），并生成该信号的数值表示，通常是二进制数值，其值介于0和$2^{(N-1)}$之间，N表示输出字的位数。

ADC噪声源

虽然图1中有多个噪声源，但有一个经常被忽略，或是被过分强调，即ADC数字输出的位数。以前，人们将ADC的位数视为评断品质的终极指标，认为16位转换器比14位转换器好出4倍。[5] 但在现在的高分辨率转换器中，位数几乎可以忽略。注意，信号链设计要奉行一条一般原则：

“某一级的输入噪声应在一定程度上低于前一级的输出噪声。”

与信号链一样，ADC内部通常也有一个噪声源占主导。所以，如果对N位ADC应用无噪声信号：

- 要么得出单个输出代码，要么得出两个相邻的输出代码，然后量化噪声占主导地位。信噪比(SNR)不会大于(6.02 N + 1.76) dB。[6]
- 多个输出代码呈高斯分布，热噪声源占主导地位。SNR不会大于：

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

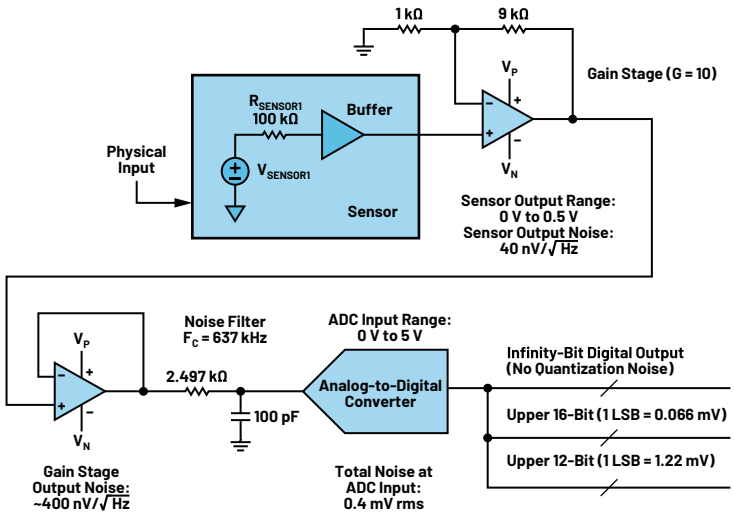

图1. 在混合模式信号链中，会将一些物理现象（例如温度、光强度、pH值、力或扭矩）转换为电气参数（电阻、电流或直接转换为电压）。然后，该信号被放大，受到低通滤波，然后被ADC数字化，ADC中可能包含内部数字滤波。

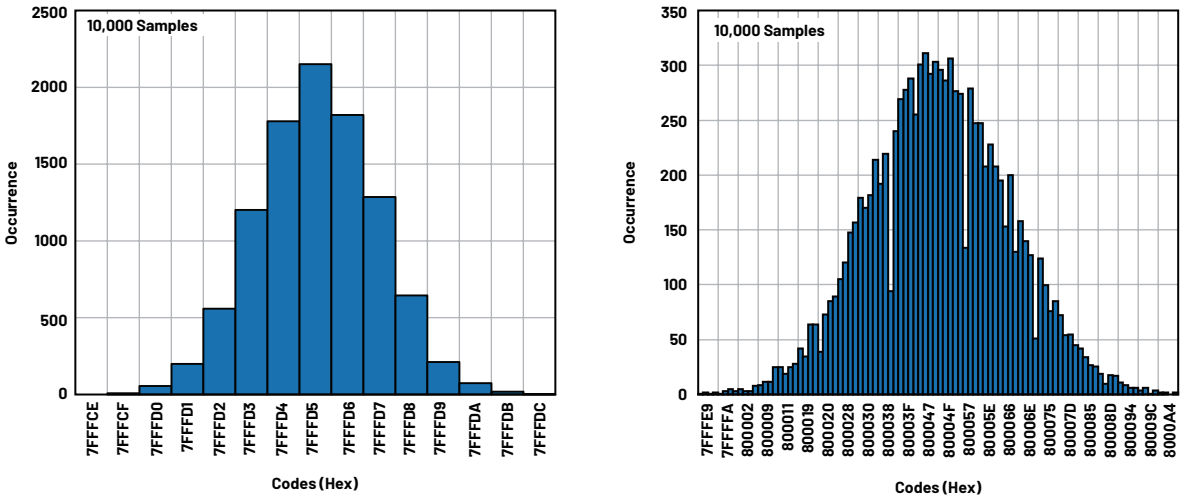

图2. 在PGA增益为1时（左侧），AD7124输出噪声中显示13个代码，标准偏差为约2.5个代码。当量化噪声可见时，热噪声更为显著。在PGA增益为128时（右侧），显示187个代码，量化噪声是无关紧要的。截断一个或两个最低有效位（双倍或四倍量化噪声）不会导致信息丢失。

其中：

V_{IN} (p-p)表示满量程输入信号。

σ表示以电压为单位的输出代码的标准偏差。

分辨率很高的转换器（例如AD7124-8，稍后会用作示例）很少受量化噪声限制；在所有增益/带宽设置中，热噪声占主导地位，短路输入始终会导致产生按高斯分布分布的输出代码。图2显示24位Σ-Δ ADC AD7124-8的接地输入直方图，内部可编程增益放大器(PGA)分别设置为1和128。

模拟和测量ADC噪声

模拟热噪声受限的ADC的噪声是很简单的。如果噪声“表现正常”（如图2所示，呈高斯分布），且在ADC的输入范围内保持恒定，即可使用NumPy[7]的随机正常函数来模拟ADC的时域噪声，然后通过标准偏差来进行验证，如图3所示。

```
# 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)
```

图3. 使用NumPy模拟高斯噪声。

AD7124设备驱动器在行业标准IIO框架之内，该框架具有完善的软件API（包括Python捆绑）。应用代码可以在本地（在树莓派上）运行，也可以通过网络、串行或USB连接在远程机器上运行。此外，pyadi-iio[8]抽象层负责与IIO器件进行连接所需的大部分样板的设置，极大地简化了软件接口。图5显示如何打开AD7124-8的连接，进行配置，捕捉一个数据块，然后关闭连接。

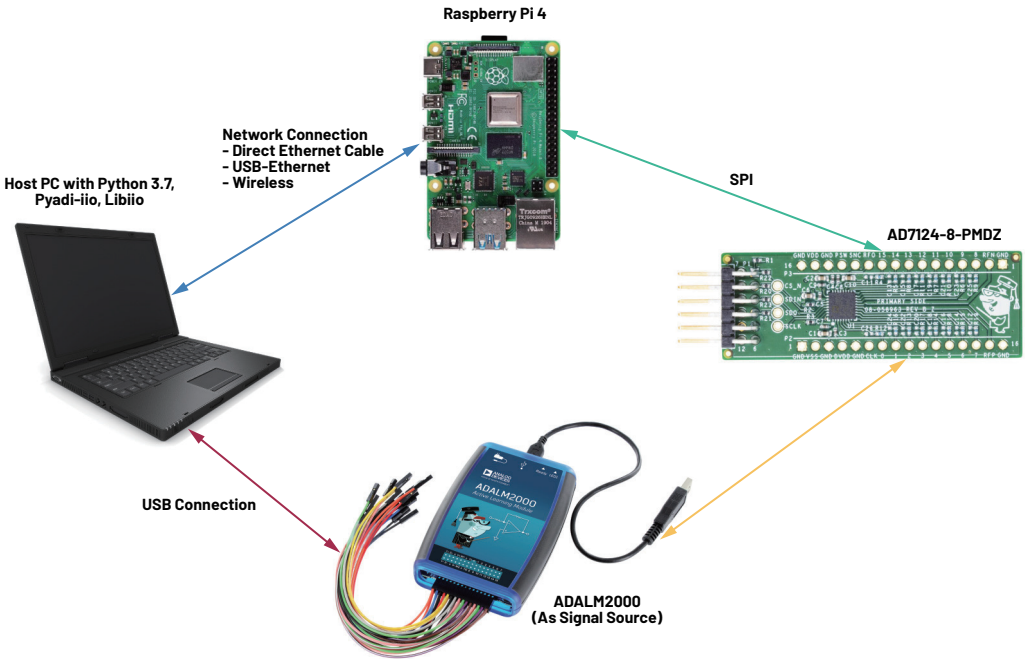

图4. ADALM2000是一款多功能USB测试仪器，具有两个通用模拟输入和两个输出，采样率分别为100 MSPS和150 MSPS。它可以作为简单的信号源，用于测量ADC噪声带宽和滤波器响应。运行支持AD7124器件驱动器支持的内核的树莓派4作为AD7124和主机之间的简单桥梁。

```
# 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
```

图5. AD7124-8基本数据捕捉。

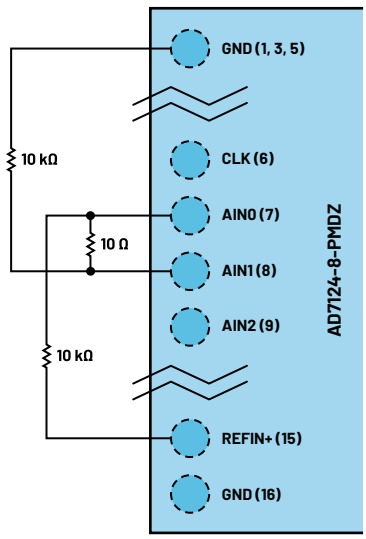

图6. 使用一个电阻分压器在AD7124-8的输入中生成1.25 mV偏置，克服15 µV最大失调电压，确保ADC的读数始终为正。

建立与AD7124-8的通信之后，可以执行非常简单，但非常有用的测试：直接测量输入噪声。简单地让ADC的输入短路，然后查看ADC代码的分布，这是确定信号链设计的一个非常有用的步骤。AD7124的输入模式设置为单极性，所以只有正值是有效的；图6所示的测试电路确保输入始终为正值。

图7显示两个1024点的测量值。下方的（蓝色）线条是在初次通电后立即获取的。

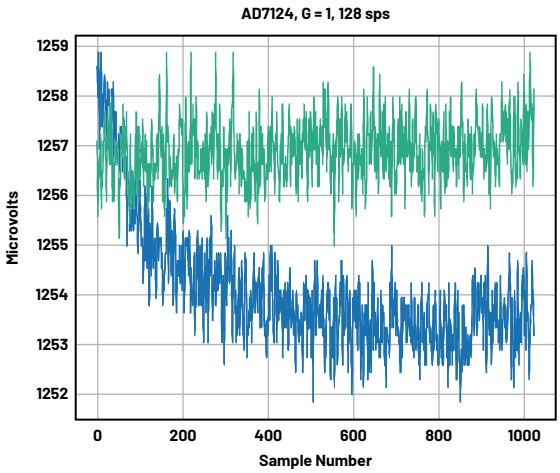

图7. 两次AD7124-8数据捕捉是在采用1.25 mV偏置的情况下进行的。下面的线条显示在通电后，电路升温时的初始漂移。上面的线条显示在半个小时升温后，读数达到稳定。

“漂移”可能是由许多因素造成的——内部基准电压源升温、外部电阻升温（因此漂移），或者是因为寄生热电偶，在热电偶中，稍微不同的金属会在存在热梯度的情况下产生电压。升温后测量到的噪声为约565 nV rms，与数据手册中的噪声规格相当。

用密度表示ADC噪声

如果所有元件都包括噪声密度规格（大部分明确规定的传感器和几乎所有的放大器都如此要求），模拟信号链设计的一般原则（某一级的输入噪声应在一定程度上低于前一级的输出噪声）将是一项简单的计算。

与放大器和传感器不同，ADC数据手册通常不包括噪声密度规格。用密度表示ADC的噪声之后，可以直接与模拟信号链的最后一个元件的输出噪声进行比较，它可能是ADC驱动器级，是增益级，或是传感器本身。

ADC的内部噪声必然会出现在DC和采样率的一半之间。理想情况下，该噪声是扁平的，或者至少是可预测的形状。事实上，由于ADC的总噪声分布在已知带宽上，所以可以将其转换成噪声密度，然后直接与信号链的其他元件进行比较。精密转换器的总噪声通常会直接给出，单位为Vrms：

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

其中e_{RMS}表示总有效值噪声，根据代码的接地输入直方图的标准偏差进行计算。

用正弦信号测试和表征的更高速度的转换器通常包含SNR规格。如果提供，可使用以下公式计算总有效值噪声：

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

其中ADCp-p是ADC的峰峰值输入范围。

可以使用以下公式计算等效噪声密度：

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

其中f_S表示ADC采样速率，单位为样本/秒。

在128 SPS的数据速率下，在升温后图7的总噪声为565 nV。噪声密度约为：

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

ADC现在可以直接纳入信号链噪声分析中，为优化信号链增益提供了指导。

- 增加增益，只要到达“ADC之前的最后一个级的噪声密度比ADC的噪声密度高一位”的点，即停止。切勿再增加信号链增益——这只会放大噪声，并减小允许的输入范围。

这与“填补”ADC的输入范围的传统智慧背道而驰。如果ADC的转换函数中存在步进或断续，超出ADC的输入范围可能会有好处，但对于“表现正常”的ADC（大多数Σ-Δ ADC和现代的高分辨率逐次逼近寄存器(SAR) ADC）来说，通过噪声进行优化是首选方法。

测量ADC滤波器响应

AD7124-8是一个Σ-Δ ADC，其中调制解调器产生高采样率，但噪声大（低分辨率），表示模拟输入。这些噪声很大的数据然后被内部数字滤波器过滤，产生更低速率、更低噪声的输出。滤波器的类型因ADC而异，具体由预期的最终应用决定。AD7124-8是针对精密应用的通用器件。因此，数字滤波器响应和输出数据速率是高度可配置的。虽然数据手册中明确定义了滤波器响应，但有时可能需要测量滤波器对给定信号的影响。AD7124-8滤波器响应代码块（参见图9）通过将正弦波应用到ADC输入并分析输出来测量滤波器响应。该方法适用性高，可用于测量其他波形——子波和模拟的物理事件。ADALM2000连接至AD7124-8电路，如图8所示。

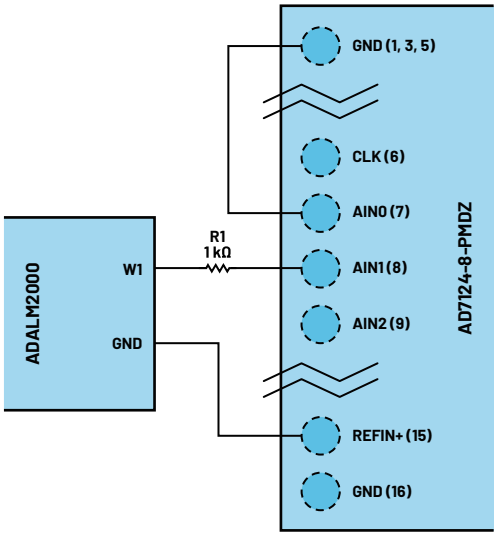

图8. ADALM2000波形发生器用于生成一定范围的正弦波频率，以直接测量AD7124-8的滤波器响应。虽然脚本将正弦波幅度和偏移设置为安全水平，1 kΩ电阻可以在功能故障时保护AD7124-8。（ADALM2000的输出电压范围为-5 V至+5 V，而AD7124-8的绝对最大限值为-0.3 V和+3.6 V。）

AD7124-8滤波器响应代码块（参见图9）将设置ADALM2000的波形发生器，生成10 Hz正弦波，捕捉1024个数据点，计算rms值，然后将结果附加到列表中。send_sinewave和capture_data是实用函数，分别用于发送一个正弦波到ADALM2000和接收来自AD7124的数据块。[2] 接着，它将频率步进增加，直到达到120 Hz，然后给出图10所示的结果。

```
# 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()
```

图9. ADALM2000的滤波器响应框图。

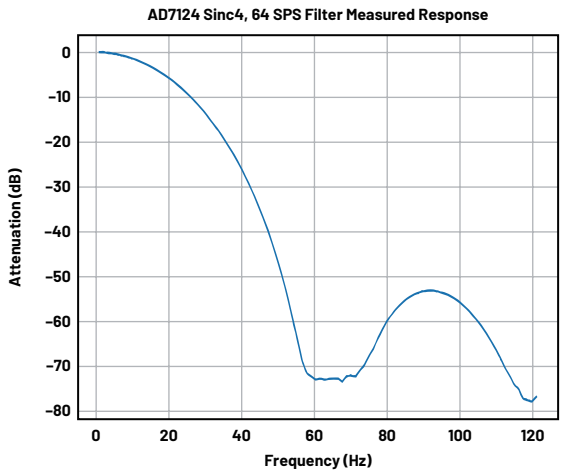

图10. 在64 SPS、sinc4模式下测量AD7124滤波器的响应，显示滤波器的通带、第一个波瓣和前两个零位。

当测量高衰减值需要一个更安静和更低失真的信号发生器时，在此设置下，前几个主要波瓣的响应是明显的。

模拟ADC滤波器

测量ADC的滤波器响应的能力是一项实用的平台验证工具。但是，要完全模拟信号链，需要滤波器的模型。关于这一点，许多转换器（包括AD7124-8）没有明确指明，但可以根据数据手册中提供的信息逆向设计得出可用的模型。

注意，以下只是AD7124-8滤波器的模型；不是位精准的表示。请参考AD7124-8数据手册查看所有保证参数。

AD7124的滤波器都具有由各种sinc函数组成的频率响应（频率响应与$(\sin\{f\}/f)^N$成正比）。这些滤波器易于构建，在零位已知的情况下可以逆向设计。

图11显示AD7124-8的10 Hz陷波滤波器。还提供高阶sinc3和sinc4滤波器的各种组合。

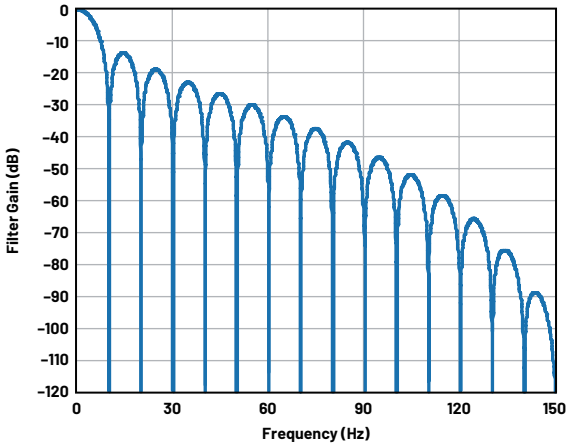

图11. AD7124-8 10 Hz陷波滤波器具有sinc1幅度响应；滤波器的脉冲响应只是100 ms时间间隔内样本的未加权（矩形）平均值。

图12中显示的同步50 Hz/60 Hz拒波滤波器是一个重要示例。此滤波器用于强烈抑制来自交流电源线的噪声，可能是50 Hz（与欧洲一样），或者是60 Hz（与美国一样）。

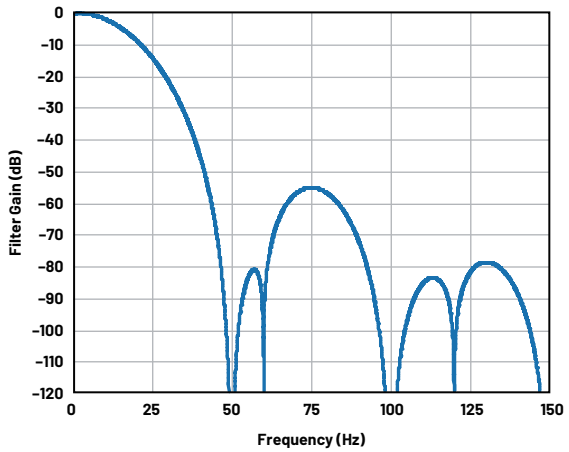

图12. AD7124-8 50 Hz/60 Hz拒波滤波器响应是50 Hz sinc3滤波器和60 Hz sinc1滤波器的组合。

可以通过对sinc1滤波器进行卷积来生成更高阶的sinc滤波器。例如，将两个sinc1滤波器（在时间上有一个矩形脉冲响应）进行卷积将得到一个三角脉冲响应和一个相应的sinc2频率响应。AD7124滤波器代码块（参见图13）生成一个sinc3滤波器，在50 Hz时为零，然后添加第四个滤波器，在60 Hz时为零。

```
# 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)
```

图13. 适用于50 Hz/60 Hz sinc滤波器的AD7124-8代码示例。

滤波器的脉冲（时域）形状如图14所示。滤波器系数(tap)值被标准化，以得出零频率时的单位(0 dB)增益。

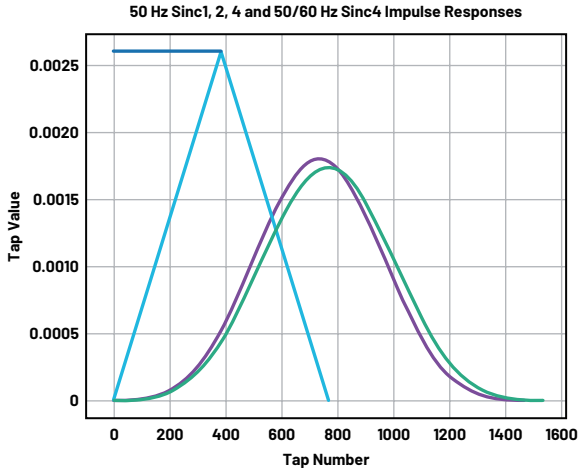

图14. 对矩形脉冲响应进行反复卷积，得到三角形响应，然后是类高斯脉冲响应。

最后，可以使用NumPy的freqz函数计算频率响应，如图16所示。响应如图15所示。

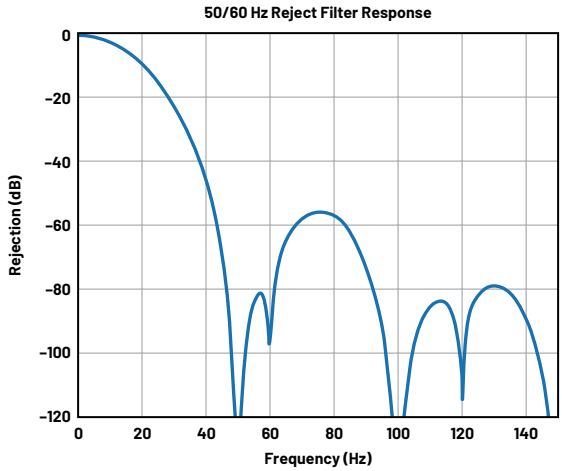

图15. 将sinc3 50 Hz陷波滤波器与sinc1 60 Hz滤波器进行卷积，将产生强烈抑制50 Hz和60 Hz的复合响应。

```
# 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)
```

图16. AD7124-8代码示例，适用于带sinc 60 Hz滤波器的sinc3 50 Hz陷波滤波器。

无可避免：传感器的基本限制

所有传感器，无论多么完美，都有最大输入值（和对应的最大输出，可能是电压、电流、电阻，甚至是刻度位置）和一个有限的本底噪声——即使输入完全静止，输出也存在“波动”。在有些情况下，提供电力输出的传感器可能包含具有有限电阻（更广泛一点，阻抗）的元件，在图17中，R_{SENSOR}表示该电阻。

这代表一个无法改善的基本噪声限值，此电阻会生成en(RMS)噪声电压，最小值为：

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

其中：

e_N(RMS)表示总噪声。

K表示波尔兹曼常数(1.38^{e-23} J/K)。

T表示电阻的绝对温度（开氏度）。

F2和F1表示相关频段的上限和下限。

将带宽标准化至1 Hz，以V/√Hz为单位表示噪声密度。

传感器数据手册可能给出低输出电阻（通常接近0 Ω），但这可能是个缓冲级，可以简化与下游电路之间的连接，但无法消除信号链前面部分的电阻导致的噪声。

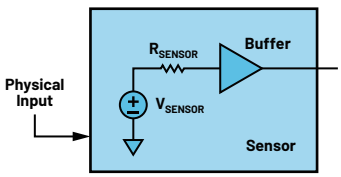

图17. 传感器通常包括一个内部缓冲器，用于简化与下游电路的连接。当输出阻抗很低（通常接近0 Ω）时，来自高阻抗检测元件的噪声与信号一起被缓冲。

还有许多其他的传感器限制——机械的、化学的、光学的，每个传感器都有自己的理论限制，我们可以模拟其影响，之后再进行补偿。但噪声是唯一无法弥补的缺陷。

实验室噪声源

一个校准过的噪声发生器就像是“世界上最糟糕的传感器”，它模拟传感器的噪声，但实际上不做任何检测。这种发生器允许直接测量信号链的噪声响应。图18所示的电路使用1 MΩ电阻作为127 nV/√Hz（在室温下）噪声源，具有“合格”的精度和带宽。虽然精度只是合格，此方法也有其优势：

- 它基于第一原则，因此在某种意义上可以作为一种未校准的标准。
- 它是真正随机的，不含重复的模式。

OP482是一款超低偏置电流放大器，具有相应的低电流噪声，以及足够低的电压噪声，所以，1 MΩ输入阻抗导致的噪声占主导地位。配置增益为2121，输出噪声为269 μV/√Hz。

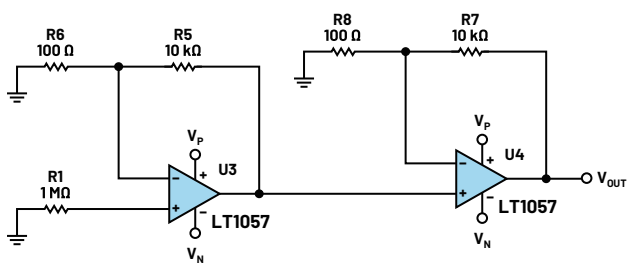

图18. 一个1 MΩ电阻作为可预测的噪声源，然后通过低噪声运算放大器放大到可用的水平。

使用ADALM2000 USB仪器，以及Scopy GUI的频谱分析仪验证噪声源，如图19所示。[9]

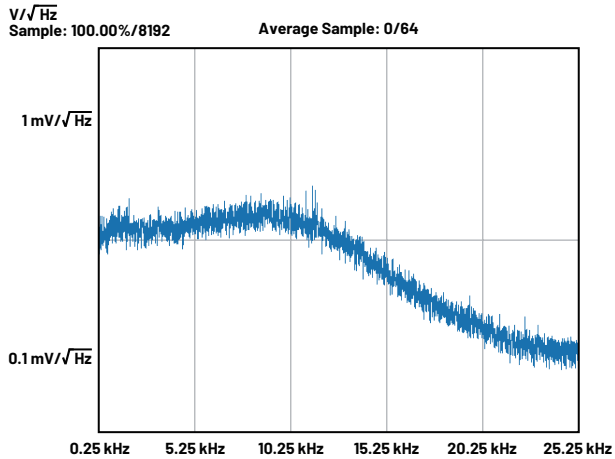

图19. 基于电阻的实验室噪声发生器的输出具有大约10 kHz的可用带宽。

在分析仪采用图示的设置时，ADALM2000的本底噪声为40 μV/√Hz，远低于噪声源的269 μV/√Hz。

虽然Scopy可用于单次可视测量，但其功能可以通过SciPy周期图函数轻松复制。使用libm2k[10]和Python捆绑程序从ADALM2000收集原始数据，进行最低限度的处理，以去除直流内容（否则会泄漏至低频率仓），并扩展至nV/√Hz。此方法如图20所示，适用于任何数据采集模块，只要采样速率是固定的、已知的，且数据可以格式化为电压向量。

```
# 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
```

图20. ADALM2000的Python噪声源测量代码。

我们现在有了已知的噪声源和测量该噪声源的方法，它们都可以用来验证信号链。

在LTspice中模拟信号链

Ltspice®是一款免费的通用模拟电路模拟器，可模拟信号链设计。它可以执行瞬态分析、频域分析（交流扫描）和噪声分析，分析结果可以导出并使用Python集成到混合信号模型中。

图21显示模拟噪声发生器的噪声模拟，与实验结果高度一致。使用与OP482的属性相似的运算放大器进行模拟。

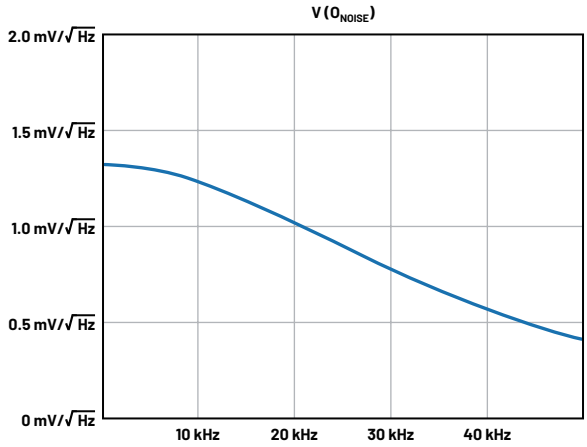

图21. 对实验室噪声源的LTspice模拟显示出与被测电路大致相同的可用带宽。

在模拟的时候，图22的电路噪声并不重要，它在某些带宽（相关信号所在的带宽）中是恒定的，而在高于这些带宽的带宽中，它会按约一阶低通响应降低。由于高阶模拟滤波或有源元件本身，这种技术在模拟非平坦本底噪声时非常有用。自动归零放大器（例如LTC2057）中常见的噪声山形就是一个典型示例，请参见图23。

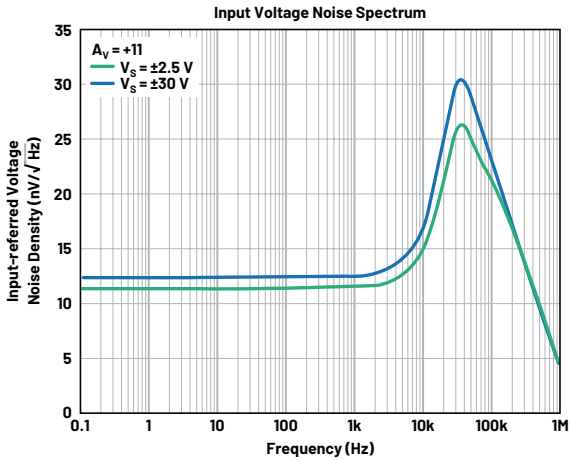

图22. LTC2057的噪声密度在低频率下是平坦的，在50 kHz时出现峰值（内部振荡器的100 kHz频率的一半）。

在Python中导入LTspice噪声数据用于频域分析涉及到设置模拟命令，以模拟分析向量中的具体频率。在本例中，噪声模拟的最大频率设置为2.048 MHz，分辨率为62.5 Hz，对应于4.096 MSPS采样率下的第一奈奎斯特区。图23显示同相增益为10时对LTC2057的模拟、模拟输出和导出的数据格式。

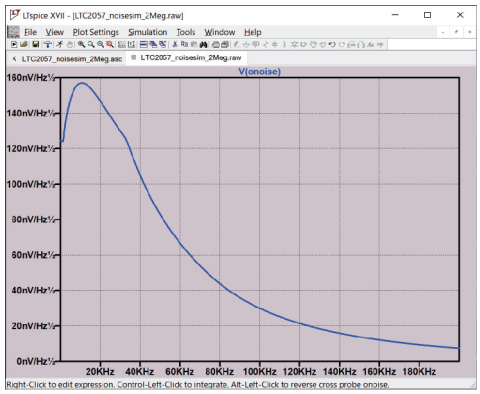

图23. LTspice用于模拟LTC2057在同相增益配置为+10时的输出噪声。LTspice提供了用于集成噪声的简单工具，但是可以将任何模拟的结果导出和导入到Python中，以进行进一步的分析。

为了确定给定频带的噪声对信号（信噪比）的影响，在相关带宽上集成噪声的和的平方根。在LTspice中，可以通过设置绘图界限来集成绘制参数，然后单击参数标签。整个2.048 MHz模拟过程的总噪声为32 µV rms。在Python中实现此操作的函数如图24所示。

```
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
```

图24. 用于实现和的平方根的Python代码。

读取导出的噪声数据并将其传递给integrate_psd函数，得出的总噪声为3.21951e-05，与LTspice计算得出的值非常接近。

生成测试噪声

它在纯粹的模拟噪声发生器的功能上进行扩展，非常适合用于生成不止是扁平，而且是任意的噪声剖面——平坦的噪声带、粉红噪声，或模拟某些放大器的峰值的噪声山形。由图25所示的半谱代码块生成的时间序列从所需的噪声谱密度（可以手动生成，或从LTspice模拟中获取）和时序序列的采样速率开始，然后生成可以发送至DAC的电压时间序列值。

```
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))
```

图25. 生成任意噪声剖面的Python代码。

可以通过使用libm2k脚本控制一个ADALM2000，然后使用第二个ADALM2000和Scopy GUI中的频谱分析仪来验证噪声剖面，以验证此功能。将噪声时间序列推到ADALM2000代码片段（参见图26），会在ADALM2000 W2输出上生成4个1 mV/√Hz噪声带（在W1上有一个正弦波，用于实现双重检查功能）。

```
# 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)
```

图26. 使用ADALM2000验证任意噪声。

图27显示了一个ADALM2000生成的4个1 mV/√Hz噪声带。输入矢量长达8192个点，采样速率为75 kSPS，每个点带宽为9.1 Hz。每个频段为512个点，或为4687 Hz宽。高于~20 kHz之后出现的滚降是DAC的sinc滚降。如果DAC能够提供更高的采样速率，时间序列数据就可以通过插值滤波器进行上采样和滤波。[11]

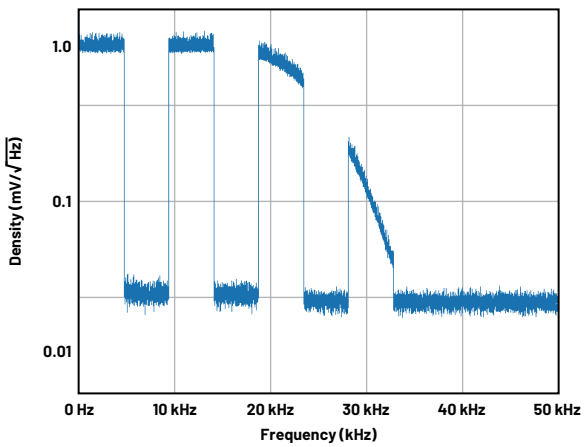

图27. Scopy光谱分析仪被用于验证任意噪声发生器。噪声带之间的深凹痕展示了分析仪的本底噪声，表明可以准确地生成任意噪声剖面。

该噪声发生器可与纯粹的模拟发生器一起使用，用于验证信号链的抑制特性。

模拟和验证ADC噪声带宽

外部噪声源和$f_S/2$以上的杂散音将折回（混叠）到直流到$f_S/2$区域，转换器可能对远远超过$f_S/2$的噪声非常敏感。以LTC2378-20为例，它具有1 MSPS采样速率，34 MHz的-3 dB输入带宽。虽然在如此高的频率下性能可能不是最好的，但这个转换器会对超过68个奈奎斯特区的噪声进行数字化，并将它们折叠回您的信号上。这展示了抗混叠滤波器对宽带ADC的重要性。精密应用的转换器一般采用Σ-Δ（例如AD7124-8）或过采样SAR架构，在该架构中，输入带宽受设计限制。

考虑滤波器的等效噪声带宽(ENBW)通常是有用的，包括ADC的内置滤波器。ENBW是扁平通带“砖墙”滤波器的带宽，该滤波器允许通过与非扁平滤波器相同数量的噪声。常见示例包括一阶RC滤波器的ENBW，其公式为：

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

其中f_c表示该滤波器的截止频率。如果对1 kHz一阶低通滤波器的输入和1.57 kHz砖墙低通滤波器的输入应用宽带噪声（从“直流到可见光”），输出端的总噪声功率将是相同的。

图28中的ENBW示例代码块接受滤波器幅度响应，然后返回有效噪声带宽。计算并使用单极性滤波器的幅度响应来验证ENBW = $f_c \times \pi/2$关系。

```
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)
```

图28. Python代码示例，用于计算有效噪声带宽。

此函数可用于计算任意滤波器响应的ENBW，包括AD7124的内部滤波器。可以使用与之前的50 Hz/60 Hz拒波滤波器示例类似的方法来计算AD7124 sinc4滤波器的频率响应和128 SPS采样速率。arb_anbw函数返回约31 Hz的ENBW。

ADALM2000噪声发生器可用于验证这一结果。设置测试噪声发生器生成1000 μV/√Hz的频段会导致约5.69 mV rms的总噪声，测量出的总噪声约为5.1 mV rms。示波器捕获的ADC输入信号在ADC输出数据旁边绘出，如图29所示。注意，测量到的峰峰值噪声为426 mV，而ADC的峰峰值噪声约为26 mV。虽然在真实的精密信号

链中，是无法实现如此高的噪声水平的（虽然希望如此），但本练习表明，可以将ADC的内部滤波器作为信号链中的主要带宽限制元件，从而降低噪声。

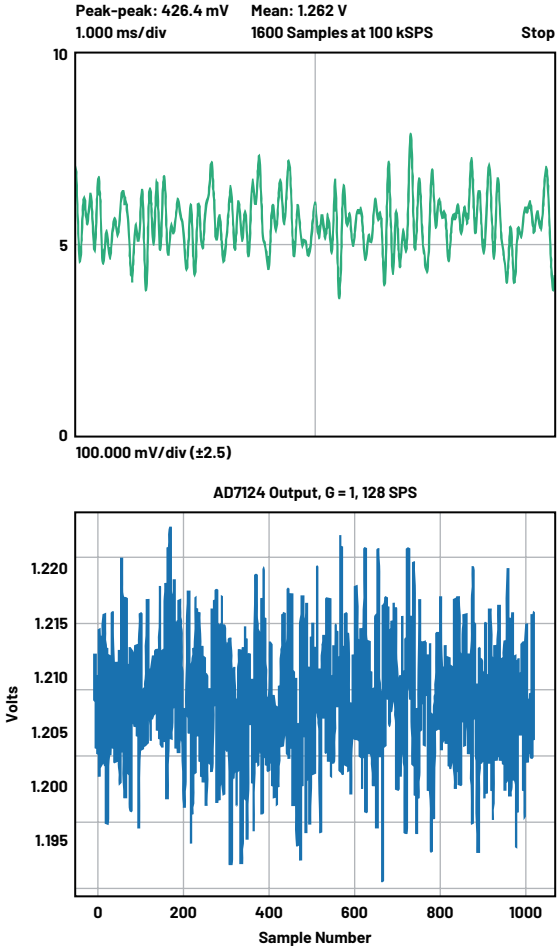

图29.1 mV/√Hz噪声带被驱动进入AD7124-8输入。很明显能够看到，噪声出现定量降低；ADC输入上的426 mV峰峰值噪声会导致ADC输出上出现约25 mV峰峰值噪声。按照给出的ADC滤波器的噪声密度为1 mV/√Hz，ENBW为31 Hz，5.1 mV rms总输出噪声非常接近预计的5.69 mV rms。

结论

在任何信号链中，噪声都是一个限制因素；一旦噪声污染信号，就会致使信息丢失。在构建信号采集系统之前，必须先了解应用要求，选择合适的组件，并测试原型电路。本教程提供一组可以在设计和测试过程中用来准确模拟和测量传感器和信号链噪声的方法。

如果单独来看，本教程中详细介绍的技术并不新颖。但是，为了实现一个合适的系统，就需要一组基本的、易于实现的低成本技术，以实现信号链模拟和验证。即使制造商继续提供性能更好的部件，但这些部件总是存在一定的限制，我们必须意识到这一点。这些技术不仅可以用于在构建混合模式信号链之前验证部件，还可以用于识别现有信号链中的设计错误。

致谢

- 感谢Jesper Steensgaard，从LTC2378-20开始，他推动了信号链设计思维范式的转变。
- 感谢Travis Collins，他架构了Pyadi-iio（还有许多其他架构）。
- 感谢软件团队经理Adrian Suciu，他推动了libm2k的开发。

参考资料

[1] “转换器连接教程。”ADI公司，维基百科，2021年1月。

[2] ADI公司教育工具库。Zenodo，2021年7月。

[3] Pauli Virtanen、Ralf Gommers等。“SciPy 1.0：在Python中进行科学计算的基本算法。”Nature Methods，17(3)，2020年2月。

[4] Steven W. Smith。*面向科学家和工程师的数字信号处理指南*。California Technical Publishing，1999。

[5] Ching Man。“MT-229：量化噪声：公式SNR = 6.02 N + 1.76的扩展推导。”ADI公司，2012年8月。

[6] Walt Kester，“MT-001：揭开公式“SNR = 6.02N + 1.76dB”的神秘面纱，以及为什么我们要予以关注。”（ADI公司，2009年）

[7] Charles R. Harris、K. Jarrod Millman等。“使用NumPy的阵列编程。”Nature，585，2020年9月。

[8] “pyadi-iio：适用于IIO驱动器的器件特定的Python接口。”ADI公司，维基百科，2021年5月。

[9] “Scopy。”ADI公司，维基百科，2021年2月。

[10] “什么是Libm2k？”ADI公司，维基百科，2021年10月。

[11] Walt Kester，“MT-017：过采样插值DAC。”（ADI公司，2009年）

归属

本文首次出现在2021年Python科学计算大会的会议记录中，题为“使用Python来分析和验证混合模式信号链”。DOI：10.25080/majora-1b6fd038-001。

作者简介

Mark Thoren于2001年加入凌力尔特（现为ADI公司的一部分），担任应用工程师，负责精密数据转换器支持工作。在此期间，他曾担任与混合信号应用相关的多个职位，包括开发评估系统、培训、刊发技术资料，以及提供客户支持。Mark现在是ADI公司系统开发团队的一名系统工程师，负责开发参考设计和ADI大学计划。Mark拥有缅因大学奥罗诺分校颁发的农业机械工程学士学位和电子工程硕士学位。联系方式：mark.thoren@analog.com。

Cristina Șuteu于2019年加入ADI公司的系统开发团队，担任系统应用工程师。在ADI公司工作期间，她致力于改善软件，分别担任过技术刊物发行及培训等多个不同职位，并为ADI大学计划提供学习教材，包括ADALM2000视频系列。Cristina拥有罗马尼亚布加勒斯特大学颁发的电子工程学士学位，以及布加勒斯特大学和法国波尔多大学联合颁发的信号和图像处理硕士学位。联系方式：cristina.suteu@analog.com。

在线支持社区

ADI EngineerZone™ 中文技术论坛

访问ADI在线支持社区，与ADI技术专家互动。提出您的棘手设计问题、浏览常见问题解答，或参与讨论。

请访问ez.analog.com/cn

ANALOG DEVICES 超越一切可能™

TA23533sc-3/22

请访问analog.com/cn