AN018 — 用过采样和求均值提高 ADC 分辨率

相关器件

本应用笔记适用于下列器件：

C8051F000、C8051F001、C8051F002、C8051F005、C8051F006、C8051F010、C8051F011、C8051F012、C8051F015、C8051F016 和 C8051F017。

1. 引言

很多应用都需要使用模/数转换器（ADC）进行测量。这些应用所需要的分辨率取决于信号的动态范围、必须测量的参数的最小变化和信噪比（SNR）。因此，很多系统使用较高分辨率的片外ADC。然而也可以通过使用一些技术来达到较高的分辨率和SNR。本应用笔记介绍用过采样和求均值的方法来提高模/数转换的分辨率和SNR。过采样和求均值技术可以在不使用昂贵的片外ADC的情况下提高测量分辨率。

本应用笔记讨论如何使用过采样和求均值的方法来提高模/数转换（ADC）测量的分辨率。另外，本文最后的附录A、附录B和附录C分别给出了对ADC噪声的深入分析、最适合过采样技术的ADC噪声类型及使用过采样和求均值技术的示例代码。

2. 关键点

- 可用过采样和求均值技术提高测量分辨率，避免采用昂贵的片外ADC。
- 过采样和求均值技术对SNR和测量分辨率的改善是以增加CPU时间和降低数据通过率为代价的。
- 对于白噪声的情况，过采样和求均值可以改善信噪比。

数据转换器噪声源

ADC 转换时可能引入很多种噪声。例如：热噪声、散粒噪声、电源电压变化、基准电压变化、由采样时钟抖动引起的相位噪声以及由量化误差引起的噪声。由量化误差引起的噪声通常被称为量化噪声。这些噪声源的噪声功率是可以变化的。有很多技术可用于减小噪声，例如：改进电路板设计和在基准电压信号线上加旁路电容。但是 ADC 总是存在量化噪声，所以一个给定位数的数据转换器的最大 SNR 由量化噪声（不使用过采样技术时）定义。在正确的条件下，过采样和求均值会减小噪声和改善 SNR，这将有效地提高测量分辨率的位数。图 1 所示的系统可以用 Cygnal 的片内 ADC 和一个软件程序来实现，软件程序先采样一组样本，然后求这些样本的平均值（滤波）而得到结果。

Silicon Laboratories Inc.
4635 Boston Lane
Austin, TX 78735
Email: mcuinfo@silabs.com
Internet: www.silabs.com

新华龙电子有限公司
深圳市福田区华强北路现代之窗大厦 A 座 13F C 室(518013)
电话：0755-83645240 83645242 83645244 83645251
电邮：shenzhen@xhl.com.cn
网址：www.xhl.com.cn

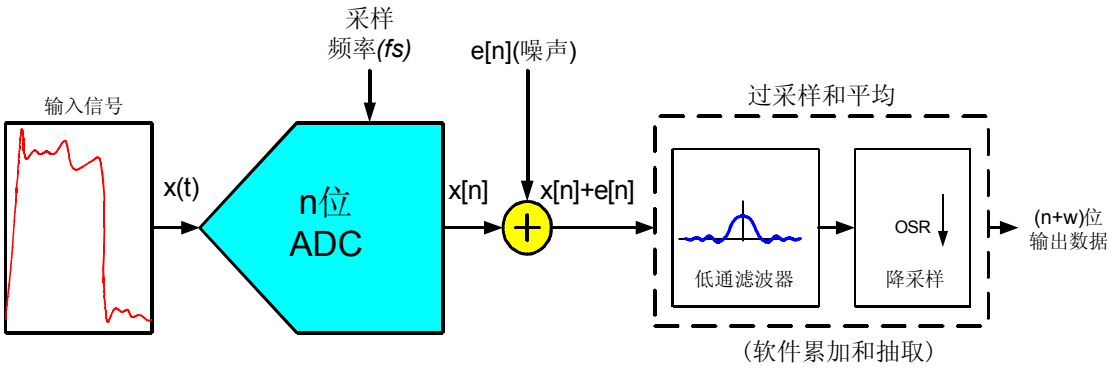

图 1. 用过采样和求均值使测量分辨率增加“W”位

3. 提高 ADC 测量的分辨率

很多应用需要测量大动态范围的信号值，还可能需要用高分辨率测量某个参数的微小变化。例如，ADC 要测量很大的温度范围，还要求系统对小于 1 度的变化作出响应。这样的系统可能需要 16 位的测量分辨率。使用 Cygnal 的片内 12 位 ADC 并采用过采样和求均值技术即可达到以 16 位分辨率进行参数测量的目的，而不必使用昂贵的片外 16 位 ADC。

某些应用要使用 ADC 分析带有高频成分的信号，这样的系统也会从过采样和求均值技术受益。根据奈奎斯特定理，所要求的采样频率为奈奎斯特频率：

$$f_n = 2 \cdot f_m$$

其中，f_m是输入信号的最高频率

方程 1. 奈奎斯特频率

采样频率（f_s）高于f_n则为过采样，过采样能提高测量分辨率。关于过采样技术提高测量分辨率的原理请参见附录A。

3.1 根据要增加的分辨率计算过采样比

为了增加有效位数（ENOB），信号被过采样，或者说ADC以高于系统所需采样频率f_s的速率对信号采样。所需要的采样频率由系统对参数测量所要求的频度（输出数据字的速率）决定，或者是奈奎斯特频率f_n。

每增加一位分辨率，信号必须被以 4 倍的速率过采样：

$$f_{os} = 4^w \cdot f_s$$

其中，w 是希望增加的分辨率位数，f_s是初始采样频率要求，f_{os}是过采样频率。

方程 2. 增加测量分辨率的过采样频率

附录 A 中介绍了方程 2 的一个导出方程。

假设一个系统使用 12 位的 ADC，每秒输出一个温度值（1Hz）。为了将测量分辨率增加到 16

位，我们按下式计算过采样频率：

$$f_{os} = 4^4 \cdot 1(Hz) = 256Hz$$

因此，如果我们以f_s=256Hz的采样频率对温度传感器进行过采样，我们将在所要求的采样周期内采集到足够的样本，对这些样本求均值便可得到 16 位的输出数据。为此，我们先累加（将 256 个连续样本加在一起），然后将总和除以 16（或将总和右移 4 位）。这样的过程通常被称为*抽取*。这样得到的结果是 16 位的有用数据，所做的操作被称为*累加和抽取*。一旦我们计算得到由 256 个样本（对本例而言）所产生的结果，我们将对数据进行保存或处理，然后开始为下一个输出字采集样本。

注：用于累加过采样数据和进行除法运算的存储器单元所占的字节数必须足够多，以免发生溢出或产生截断错误。

附录 C 中给出了一个过采样和求均值的例子。在该例中，用片内 12 位 ADC 对片内温度传感器采样，得到 16 位的测量结果。有关过采样对噪声的影响和提高分辨率的更深入的讨论见附录 A。

3.2 根据要增加的 SNR 计算过采样率

在不进行过采样和求均值时，ADC 测量的 SNR 理论极限是由模数转换过程中固有的量化噪声决定的。因为量化误差取决于 ADC 的分辨率位数（见方程 5），所以最好情况下的 SNR 值是数据转换有效位数的函数，计算公式如下：

$$SNR(db) = (6.02 \cdot ENOB) + 1.76$$

其中，ENOB 是测量值的有效位数。

方程 3. SNR 为 ENOB 的函数

注意，方程 3 对*满度*输入有效，这就是说输入信号的动态范围必须与 ADC 的基准电压一致。否则实际 SNR 比用方程 3 计算出来的值要低。

如果用于测量某个参数的 ADC 是 12 位的并且不采用过采样技术，则最佳 SNR 值（方程 3 计算）为 74dB。如果我们想得到更高的 SNR，则必须根据给定的 SNR 用方程 3 计算所需要的 ENOB。一旦我们知道所要求的 ENOB，即可用方程 2 计算所需要的过采样频率。

例如，如果一个应用所要求的 SNR 为 90dB，则我们至少需要 16 位的分辨率。使用一个 12 位的 ADC 并根据方程 2 计算，得知必须以 256 倍的频率进行过采样。

3.3 过采样和求均值法的有效性

过采样和求均值法的有效性取决于主要噪声源的特性。最关键的要求是噪声源应为白噪声。请见附录 B 中对于能用过采样和求均值法改善的噪声源特性的讨论。要考虑的关键点是：

- 噪声必须逼近白噪声，在整个有用频带内具有平均分布的功率谱密度。
- 噪声幅度必须足够大，能引起输入信号样本之间的随机变化，变化幅度至少为两个相邻代码之间的距离（即一个 LSB，见附录 A 中的方程 5）。
- 输入信号可以用一个在两个相邻 ADC 代码之间具有等概率分布的随机变量表示。

注：过采样和求均值不能补偿 ADC 的积分非线性误差（INL）。

过采样和求均值技术对相关或不能用白噪声建模的噪声（例如，反馈系统的噪声）不起作用。另外，如果量化噪声的功率大于自然白噪声（例如热噪声），过采样和求均值技术也不会奏效。ADC 的分辨率较低时就属于这种情况。大多数使用 12 位 ADC 的应用都可以从过采样和求均值技术获益。

有关这一问题的更深入的讨论请见附录 B。

4. 实例

本应用笔记的附录 C 中给出了一个使用过采样和求均值技术的例子。该程序使用 Cygnal 的片内、100ksps、12 位 ADC 对片内温度传感器执行 16 位精度的测量，从硬件 UART 输出数据。

根据方程 2，过采样比为 256。所提供的代码（在“AN018_SW.c”中）将 256 个连续的 ADC 样本累加到变量 *accumulator*。在完成累加后，又将 *accumulator* 右移 4 位并将结果存入变量 *result* 中。在得到计算结果后 *accumulator* 被清空（清 0），准备进行下一次计算。对 ADC 样本的累加是在 ADC 转换完成中断服务程序（ADC_isr）中完成的。

有关配置和使用片内温度传感器的更详细的信息见应用笔记“AN003 - 使用片内温度传感器”。

4.1 分辨率改善

我们使用过采样和求均值技术将对温度传感器的测量精度从 12 位提高到 16 位。下面我们对温度测量中的分辨率改善情况进行比较。

片内温度传感器的满度输出略大于 1 伏。假设使用 2.4V的基准电压（V_{ref}），我们可以计算 12 位和 16 位测量的代码宽度和温度分辨率（可测量的最小温度变化）。

12 位温度分辨率

在不采用过采样技术的情况下，我们将得到 12 位的温度测量结果。温度每变化一个摄氏度，片内温度传感器的电压将变化 2.8mV。在使用 2.4V的V_{ref}且PGA增益等于 2 时，电压分辨率是（使用附录A中的方程 5）：

$$\Delta = \frac{2.4}{2^{12} \cdot 2} = 293\mu V / ^{\circ} C$$

Δ 是方程 5 中定义的代码宽度。分母中的因子 2 是考虑到 PGA 的增益为 2。

12 位测量的温度分辨率是（每个 ADC 码所代表的摄氏度数）：

$$T_{res\,12} = \frac{293\ \mu V}{code} \cdot \frac{^{\circ}C}{2.8mV} = 0.1046\ ^{\circ}C/code$$

T_{res12}是 12 位测量的温度分辨率。

因此，对于每个 ADC 码，我们可以测量的最小温度变化是 0.104 摄氏度，或者说大于十分之一度。可能我们会需要更高的温度分辨率以便能分辨百分之一度。通过使用过采样和求均值技术，我们可以用同一个 12 位 ADC 达到这个分辨率。

16 位温度分辨率

用过采样和求均值技术使有效位数（ENOB）增加到 16 位时，新的分辨率计算如下：

$$\Delta = \frac{1.2}{2^{16}} = 18.3\mu V/^{\circ}C$$

这样，我们可以测量的最小温度变化是：

$$T_{res16} = \frac{18.3\mu V}{code} \cdot \frac{^{\circ}C}{2.8mV} = 0.0065^{\circ}C/code$$

T_{res16}是 16 位测量的温度分辨率。

在采用过采样和求均值技术的情况下，我们用同一个片内 12 位 ADC 可以测量的最小温度变化是 0.007 摄氏度。这就允许我们以高于百分之一度的精度对温度进行测量。

数据通过率降低

通过率是指每单位时间我们能得到的输出数据字的个数。如果一个ADC的最大采样速率是100ksps，在不采用过采样和求均值技术的情况下我们可以得到 100ksps的输出字速率。但是，如果我们为达到较高的分辨率而采用过采样和求均值技术（抽取），吞吐率将降低到初始值除以过采样比（OSR）（见方程 7）。在我们所提供的例子中，过采样比为 256，我们的输出字速率将是100ksps/256=390 个样本/秒（390Hz）。由此可以看出，对于给定的采样速率，应在分辨率和数据通过率之间权衡。另一个需要考虑的问题是，增加分辨率需要增加采样速率和计算时间，因此在每个采样周期（$1/f_s$）内CPU的带宽将降低。

4.2 小结

如果 ADC 噪声近似为白噪声，就可以使用过采样和求均值技术来提高 SNR 和测量分辨率。该技术既适用于静态直流测量，也适用于对含有较高频率成分的输入信号的测量。方程 2 说明，每增加一位分辨率可以通过用 4 倍的过采样频率进行采样来达到，而分辨率每增加一位将使 SNR 增加 6 dB（方程 3），这些都是以牺牲数据通过率和消耗 CPU 带宽为代价的。

附录 A — 噪声和过采样理论

本部分讨论过采样和求均值如何影响带内噪声以及如何根据所要求的 SNR 和测量分辨率计算过采样比。

过采样和求均值如何改善性能

过采样和求均值是为了完成两个任务：改善信噪比和提高有效分辨率（即增加 ADC 测量的有效位数）。这两个任务实际上是同时完成的。例如，如果我们有一个 12 位 ADC 而希望产生 16 位分辨率的转换代码，则我们用过采样和求均值技术可以得到与 16 位 ADC 相同的 SNR。这将增加测量数据的*有效位数*（ENOB），也是提高 SNR 的一种方法。过采样和均值滤波器允许我们在得到较低噪声强度的同时得到 16 位的输出字。

过采样如何影响带内噪声

采样频率f_s允许重建位于采样频率一半以内的有用信号（奈奎斯特定理）。如果采样频率为100kHz，则频率低于 50kHz的信号可以被可靠地重建和分析。与输入信号一起，还会有噪声信号混叠在有用的测量频带内（小于f_s/ 2 的频率成分）：

$$E(f) = e_{rms} \cdot (\frac{2}{fs})^{1/2}$$

其中，e_{rms}是平均噪声功率，f_s是采样频率，E(f)是带内ESD。

方程 4. 带内噪声的能量谱密度

方程 4 说明，信号频带内的噪声能量谱密度（ESD）或被采样噪声的噪声水平随采样频率的增加而降低[3]。

过采样与分辨率提高之间的关系

给定由量化噪声引起的固定噪声功率，我们可以计算增加有效分辨率所需要的过采样比。例如，我们用一个 12 位的 ADC 测量一个参数，要将测量的有效位数提高到 16 位，则我们需要建立一个能计算过采样比的公式。为此，我们首先定义噪声的特性。

噪声分析

为了理解过采样和求均值对噪声的影响，我们必须首先定义什么是量化噪声。

两个相邻 ADC 码之间的距离决定量化误差的大小。因为 ADC 会舍入到最近的量化水平或 ADC 码，所以：

$$\Delta = \frac{V_{ref}}{2^N}$$

其中，N 是 ADC 码的位数，Vref 是基准电压。

方程 5. 相邻 ADC 码之间的距离或 LSB

量化误差为（e_q）：

$$e_q \le \frac{\Delta}{2}$$

假设噪声近似为白噪声，代表噪声的随机变量在 ADC 码之间分布的平均值为零。则方差为平均噪声功率，计算如下：

$$e_{rms}^2 = \int_{-\Delta/2}^{\Delta/2} (\frac{e_q^2}{\Delta}) de = \frac{\Delta^2}{12}$$

方程 6. ADC 量化噪声的功率

我们用过采样比（OSR）来表示采样频率与奈奎斯特频率（见方程 1）之间的比较关系，定义如下：

$$OSR = \frac{fs}{2 \cdot fm}$$

其中，fs 是采样频率，fm 是输入信号的最高频率。

方程 7. 过采样比

如果噪声为白噪声，则**低通滤波器输出端**（见图 1）的带内噪声功率为：

$$n_0^2 = \int_0^{fm} e_{rms}(f)^2 df = e_{rms}^2 (\frac{2 \cdot fm}{fs}) = \frac{e_{rms}^2}{OSR}$$

其中，n_0是滤波器输出的噪声功率。

方程 8. 带内噪声功率是 OSR 的函数

方程 8 说明我们可以通过提高 OSR 来减小带内噪声功率。过采样和求均值并不影响信号功率[1]。所以我们能提高 SNR 是因为过采样能减小噪声功率但不影响信号功率。

我们可以从方程 5、6 和 8 得到下面这个反映噪声功率与过采样比和分辨率关系的表达式：

$$n_0^2 = \frac{1}{(12 \cdot OSR)} (\frac{V_{ref}}{2^N})^2$$

其中，OSR 是过采样比，N 是 ADC 的位数，Vref 是基准电压。

方程 9. 噪声功率是 OSR 和分辨率的函数

反过来，给定一个固定的噪声功率，我们可以计算所需要的位数。解方程 9 求 N，我们得到用给定的基准电压、带内噪声功率及过采样比来计算有效位数的方程 10 [1]。

$$N = -\frac{1}{2}\log(OSR)_2 - \frac{1}{2}\log(12)_2 - \frac{1}{2}\log(n_0^2)_2 + \frac{1}{2}\log(V_{ref})_2$$

方程 10. 有效位数是基准电压、带内噪声功率和过采样比的函数

从方程 10 我们可以注意到：

采样频率每增加一倍，带内噪声将减小 3 dB，而测量分辨率将增加 1/2 位[3]。

在实际应用中，我们将一个信号的带宽限制到小于 *fs* / 2，然后以某个过采样比（OSR）对该信号采样，再对采样值求平均值（或抽取）得到结果输出数据。每增加一位分辨率或每减小 6dB 的噪声，我们需要以 4 倍的采样频率进行过采样：

$$f_{os} = 4^w \cdot fs$$

其中，w 是希望增加的分辨率位数，f_s 是初始采样频率要求，f_{os} 是过采样频率

方程 11. 增加测量分辨率的过采样频率

方程 11 就是本应用笔记开始部分介绍的方程 2。如果我们使用 12 位的片内ADC，而希望得到 16 位ADC的精度，则我们需要增加 4 位分辨率。4 的 4 次幂（用方程 11）是 256，所以我们要以奈奎斯特频率的 256 倍的频率进行过采样。如果信号带宽限制在 60Hz（f_m = 60 Hz），则我们必须以 120Hz * 256 = 30.7 kHz的频率进行过采样。我们通过改善有用频带内的SNR来提高有效分辨率。提高采样频率或OSR使有用信号频带内（所有低于*fs*/2 的频率）的噪声水平降低。

量化噪声和输入信号的频谱如图 2 所示。注意，当采用过采样时，噪声频谱与输入信号频谱的重叠部分减少。因此，在不影响输入信号的情况下，低通滤波器的选择性更强，可以滤出更多的带内噪声。滤波器输出的噪声功率用方程 8 计算。这是经过过采样和平均值滤波器后已经减小的噪声强度，图 3 说明了这一点。被滤出的噪声位于*fm*和*fm* /OSR之间。如果不使用过采样技术，滤波器将不能消除这一噪声。输出被降采样（抽取）到初始的奈奎斯特频率*fn*，降采样的比率也是OSR（见图 1）。这将使输入信号的频谱与以奈奎斯特频率采样时一样，但是噪声强度降低到e_{rms}/OSR（见图 4）。

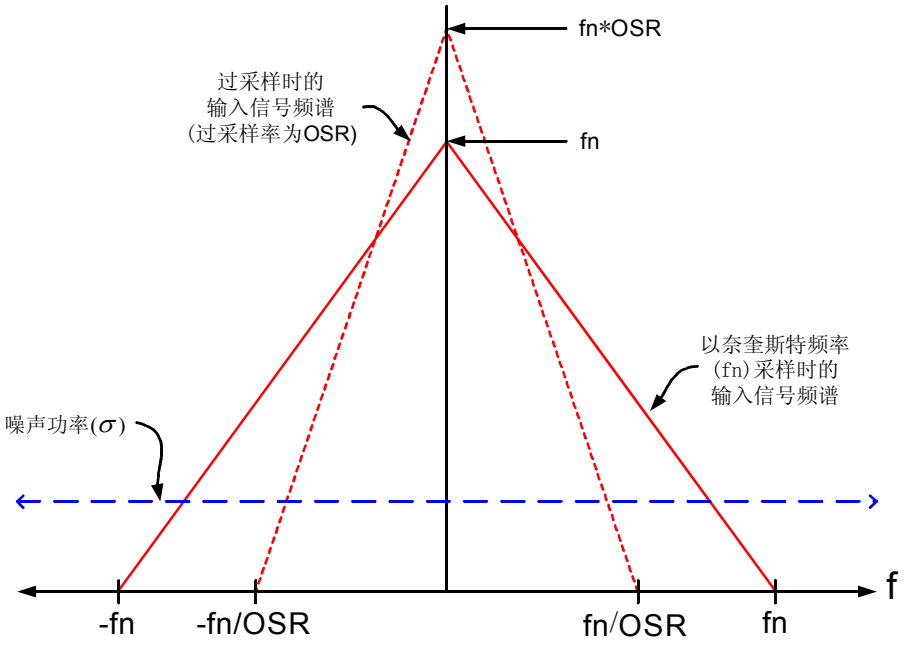

图 2. 输入信号以奈奎斯特频率和过采样频率采样时的频谱及量化噪声强度

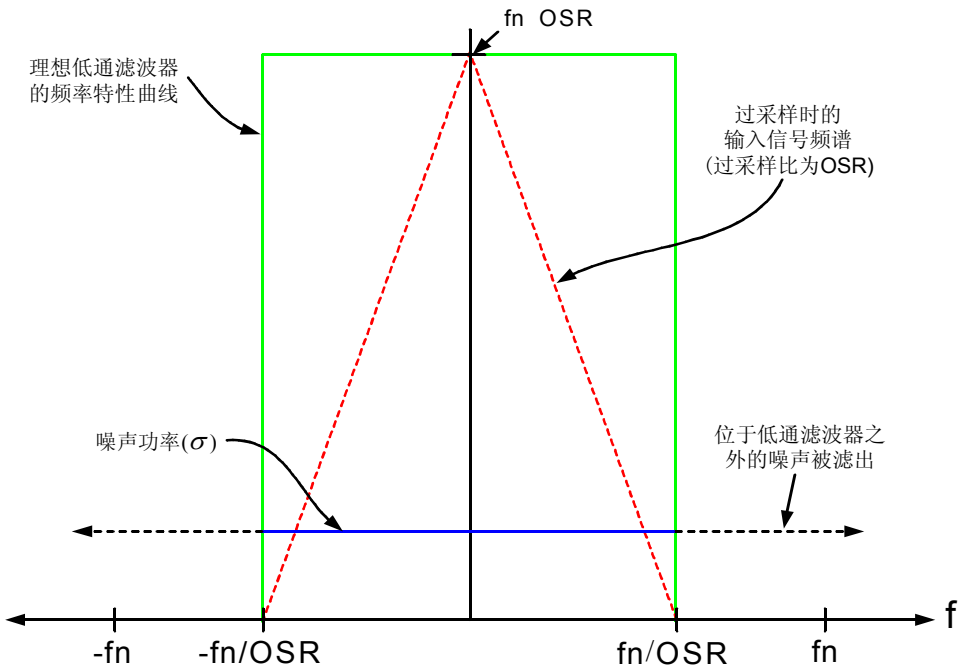

图 3. 过采样信号的频谱和理想低通滤波器对噪声的滤出

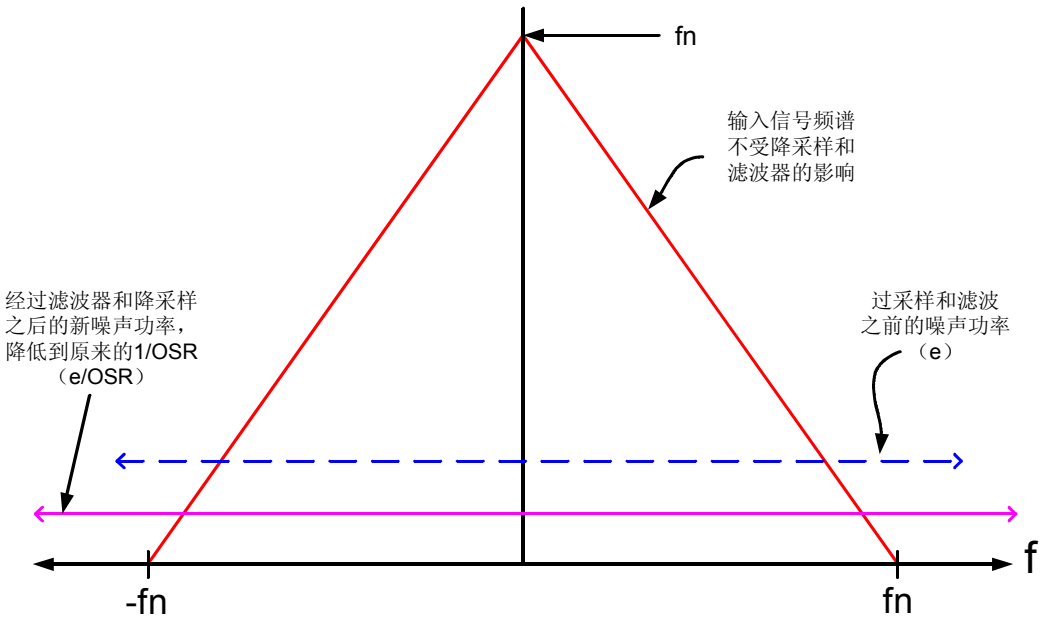

图 4. 过采样信号在经过滤波器和降采样到奈奎斯特频率后噪声功率降低

计算信噪比

信噪比被定义为信号功率有效值与噪声功率有效值的比值，以分贝（dB）表示。不管我们如何努力消除 ADC 噪声源，量化噪声总是存在的。因此，理想的 SNR 是在没有进行过采样和求均值的情况下根据量化噪声计算的。方程 5 说明：ADC 的分辨率越高，则量化误差就越小，因而量化

噪声就越低；ADC 的位数越多，则 SNR 越高。如前所述，过采样和求均值能减小带内噪声，改善 SNR 和增加有效位数（ENOB）。ENOB 是 SNR 的另一种度量形式，这两者都可用于确定技术指标和为满足这些技术指标所需要的过采样率。

为了得到最佳的SNR，输入信号的动态范围必须与参考电压（V_{ref}）一致。如果我们假设最佳情况下的输入信号是一个满度的正弦波，则它的有效值是V_{ref}的函数：

$$V_{rms} = \frac{V_{ref}}{2\sqrt{2}}$$

方程 12. 输入信号有效值是一个满度正弦波的函数

用方程 9 对噪声功率进行计算，我们得到噪声功率有效值为位数 N（未进行过采样）的函数：

$$n_0 = \frac{V_{ref}}{2^N\sqrt{12}}$$

方程 13. 噪声功率有效值

以 *dB* 表示的 SNR 计算如下：

$$SNR = 20 \cdot \log(\frac{V_{rms}}{n_0}) = 20 \cdot \log(\frac{2^N\sqrt{12}}{2\sqrt{2}})$$

方程 14. SNR 是位数的函数的函数

当使用过采样技术时，我们可以用有效位数（*ENOB*）替换方程 14 中的 *N*。化简方程 14 并用 *ENOB* 替换 *N*，我们得到下面这个非常有名的结果（以分贝表示）：

$$SNR（dB）= (6.02*ENOB)+1.76$$

其中 ENOB 是测量的有效位数

方程 15. SNR 是 ENOB 的函数

求均值增加直流测量的有效分辨率

至此，我们考虑了对位于某个有用频带内（f_m）的信号的测量。然而我们也可能测量一个相对不变的直流信号（例如温度和应变仪输出）。如果我们希望测量一个相对不变的信号（即主要频率接近直流），我们仍然可以用过采样和求均值技术改善有效分辨率[2]。

测量静态电压的应用

如果一个称重装置必须测量一个宽范围的重量，而同时又要能分辨很小的重量变化，则过采样和求均值能提高测量的有效分辨率。我们看另一个例子：ADC 必须测量一个温度传感器的输出，温度范围可能很大，但系统又要响应温度的微小变化。

过采样和求均值作为插补滤波器

对 ADC 测量数据求均值等价于一个降采样低通滤波器（见图 1）。实现过采样和低通滤波器的

数字信号处理过程通常被称为*插补*。从这个意义上说，我们用过采样实现两个 12 位 ADC 码之间的插值。求均值的样本数量越大，则低通滤波器的选择性越强，插值的效果就越好。

附录 B — 过采样和求均值何时有效

这部分讨论对于一个给定应用确定过采样和求均值是否有效的准则。

模/数转换过程引入噪声。过采样和求均值能减小某些类型的噪声，因而提高 SNR 和数据转换的有效分辨率。并非所有的应用都能从过采样和求均值受益。要理解哪些 ADC 测量能从过采样技术受益，我们必须理解给定系统中的噪声类型和特性。

有效过采样的噪声要求

过采样和求均值能改善 SNR 和提高模/数转换测量的有效分辨率。但是这一技术只在 ADC 噪声近似为白噪声的情况下有效[2] [3]。如果输入信号的样本之间以 1 个 LSB 的变化量随机变化，并且输入信号在两个相邻代码之间有相等的概率，则噪声可以被当作白噪声处理。白噪声的特点是在整个有用频带内具有一致的功率谱密度。在噪声可以被近似为白噪声的情况下，过采样和求均值可以改善 SNR 和提高数据的有效分辨率。

如果噪声在总体上不是平稳的（例如，因存在反馈而具有某种程度相关的系统），则过采样和求均值可能不会有效。另外，如果量化噪声与白噪声源相近（即热噪声和散弹噪声与量化噪声相比很小），则过采样和求均值可能不会有效。在使用较低分辨率的 ADC（例如 8 位 ADC）时这种情况比较典型。在这种情况下，热噪声没有足够的幅度能引起输入信号以等概率在相邻代码之间随机变化，因为代码宽度 Δ（方程 5）太大。某些应用会有意地在信号或处理过程中注入噪声以克服这一效应，这种处理被成为抖动。

直方图分析

大多数使用 12 位 ADC 进行测量的应用能从过采样和求均值技术中获益。一个确定噪声特性是否满足要求的实用方法是用直方图分析 ADC 输出数据（见图 5）[2]。直方图说明了在一个 ADC 结果样本集中每个代码有多少个样本。如果输入信号是一个恒定的直流电压，且噪声为白噪声，则该直方图将逼近一个高斯概率分布函数（PDF），如图 5 所示[2]。代码为 1024 的“样本子集”中的样本数最多。由于直方图逼近一个高斯 PDF（图 5 中的虚线所示），噪声逼近白噪声，因此该系统可以用过采样和求均值技术提高性能。

一个没有足够噪声幅度的系统（量化噪声除外）会导致直方图中所有样本都向一个“样本子集”或代码集中。在这样的系统中，过采样和求均值技术可能不会有帮助。

如果噪声是相关的或 ADC 的传输函数是非线性的（例如，电源噪声、INL 很差等），则直方图不能逼近一个高斯 PDF（如图 6 所示）。在这种情况下，过采样和求均值技术可能不会有帮助。

简言之，如果在 ADC 结果代码中复合噪声源逼近白噪声，样本的直方图将逼近一个高斯 PDF，过采样和求均值将改善 SNR 和提高信号测量的有效位数。

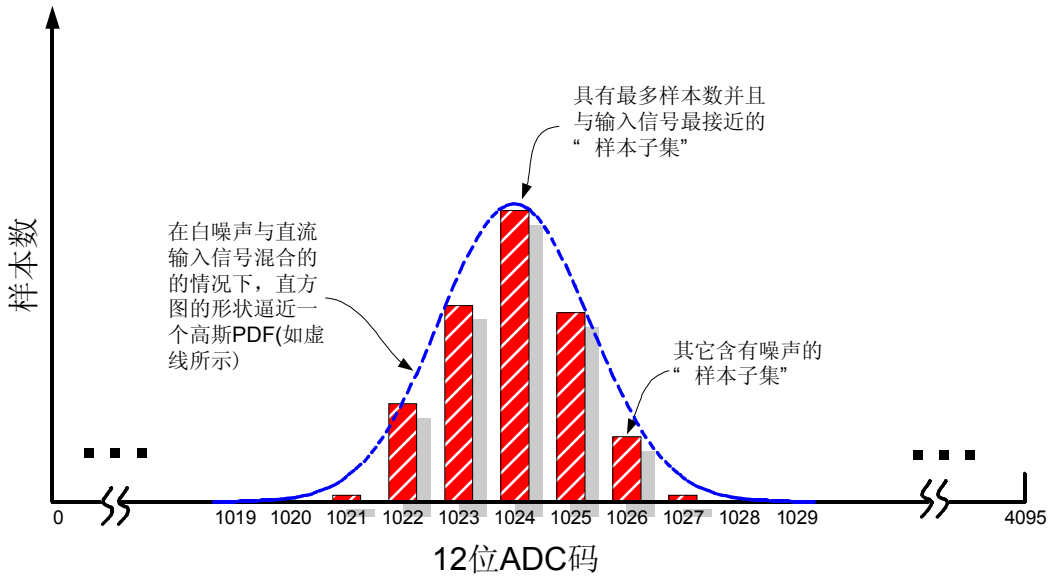

图 5. ADC 样本的直方图：混有白噪声的直流输入

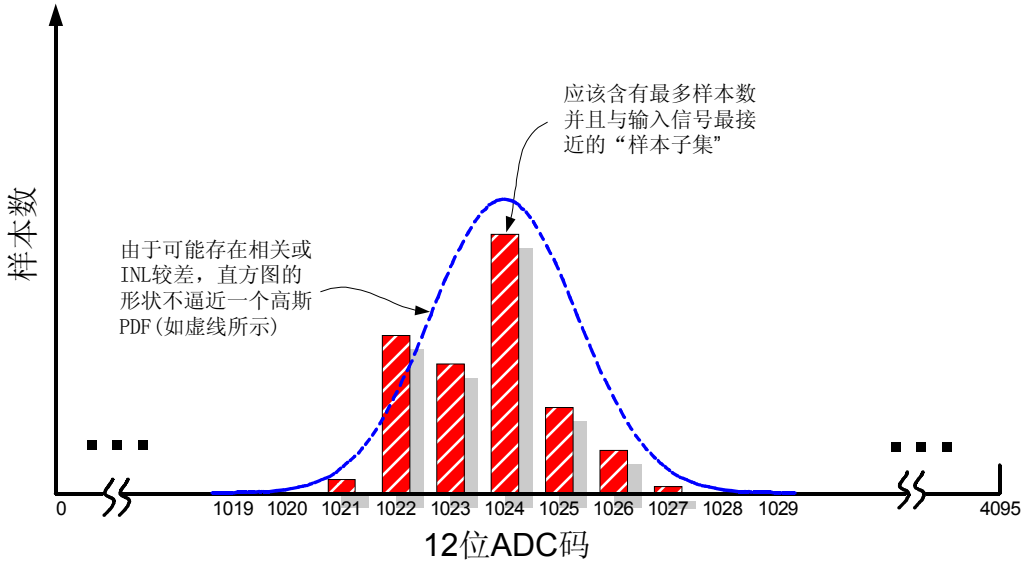

图 6. 对于过采样和求均值技术而言非最佳情况的 ADC 样本直方图

附录 C — 示例代码

```
//-----------------------------------------------------------------------
// AN018_SW.c
//-----------------------------------------------------------------------
// Copyright 2001 Cygnal Integrated Products, Inc.
//
```

```
// 作者：BW
//
// 本程序以 115.2kbps 的波特率从硬件 UART 输出 C8051Fxxx 的内部温度。假设在
// XTAL1 和 XTAL2 之间接一个 18.432MHz 的晶体。
//
// ADC 被配置为测量片内温度传感器。ADC 的采样频率由常量<SAMPLE_RATE>确定，
// 以 Hz 为单位。<SAMPLE_RATE>的最大值被限制到~86kHz，这是由所选择的
// 18.432MHz 晶体决定的(SAR 时钟 = SYSCLK / 16 = 1.152MHz。一次转换需要 16
// 个 SAR 时钟--> 72kHz 的采样速率。
//
// ADC 转换结束中断处理程序从 ADC 取出样本值并将其加到一个运行累加器中。每过 256
// 个样本 ADC 更新结果并将其存入全局变量<result>中。累加一组数值并对其进行抽取
// （每 256 个样本输出一次结果）的采样技术被称为“累加和转储”。该过程很容易实现
// 且需要很少的资源。
//
// 对于 4 的每一次幂，你可以获得一位有效分辨率。
// 对于倍数 256，你可以获得 4 位分辨率：4^4 = 256。
// 为了得到 16 位的结果，要执行右移 4 位的操作。
//
// 目标：C8051F00x 或 C8051F01x
// 工具链：KEIL C51 6.03 / KEIL C51 评估版
//

//-----------------------------------------------------------------------
// 包含的文件
//-----------------------------------------------------------------------

#include <stdio.h>
#include <c8051f000.h>                    // SFR 声明
//-----------------------------------------------------------------------
// F00x、F01x 16 位 SFR 定义
//-----------------------------------------------------------------------

sfr16 DP          = 0x82;                 // 数据指针
sfr16 TMR3RL      = 0x92;                 // 定时器 3 重载值
sfr16 TMR3        = 0x94;                 // 定时器 3 计数器
sfr16 ADC0        = 0xbe;                 // ADC0 数据
sfr16 ADC0GT      = 0xc4;                 // ADC0 下限窗口
sfr16 ADC0LT      = 0xc6;                 // ADC0 上限窗口
sfr16 RCAP2       = 0xca;                 // 定时器 2 捕捉/重载
```

```
sfr16 T2          = 0xcc;                  // 定时器 2
sfr16 DAC0        = 0xd2;                  // DAC0 数据
sfr16 DAC1        = 0xd5;                  // DAC1 数据

//-----------------------------------------------------------------------
// 全局常量
//-----------------------------------------------------------------------

#define SYSCLK      18432000          // SYSCLK 频率（Hz）
#define BAUDRATE    115200            // UART 波特率（bps）
#define SAMPLE_RATE 100000            // 采样频率（Hz）

#define LED         P1.6              // LED=1 表示亮

//-----------------------------------------------------------------------
// 函数原型
//-----------------------------------------------------------------------

void SYSCLK_Init (void);
void PORT_Init (void);
void UART_Init (void);
void ADC_Init (void);
void TIMER3_Init (int counts);
void ADC_ISR (void);

//-----------------------------------------------------------------------
// 全局变量
//-----------------------------------------------------------------------

long result;                          // 对于 16 位测量，用 ADC 进行过采样和
                                      // 对 256 个样本求均值后的输出结果。

//-----------------------------------------------------------------------
// 主程序
//-----------------------------------------------------------------------

void main (void) {
   long temp_copy;
   int temp_int;                      // 温度值的整数部分
   int temp_frac;                     // 温度值的小数部分（以百分之一度为单位）
```

```
   WDTCN = 0xde;                    // 禁止看门狗定时器
   WDTCN = 0xad;

   SYSCLK_Init ();                  // 初始化振荡器
   PORT_Init ();                    // 初始化交叉开关和 GPIO
   UART_Init ();                    // 初始化 UART
   TIMER3_Init (SYSCLK/SAMPLE_RATE); // 初始化定时器 3 以采样速率溢出
   ADC_Init ();                     // 初始化 ADC

   ADCEN = 1;                       // 允许 ADC

   result = 0L;                     // 初始化温度变量

   EA = 1;                          // 允许全局中断

   while (1) {
      temp_copy = result;           // 取最新的 ADC 结果，将 ADC 代码
                                    // 转换为温度
      temp_copy -= 0xa381;          // 将偏移量校正为 0 度对应 0V
      temp_copy *= 0x01a9;          // 2.86mV/摄氏度
      temp_copy *= 100;             // 将结果转换成百分之一摄氏度
      temp_copy = temp_copy >> 16;  // 除以 2^16
      temp_int = temp_copy / 100;   // 分离整数和小数部分
      temp_frac = temp_copy - (100 * temp_int);
    printf ("Temperature is %d.%d\n", (int) temp_int, (int) temp_frac);
    }
}

//-----------------------------------------------------------------------
// 初始化子程序
//-----------------------------------------------------------------------

//-----------------------------------------------------------------------
// SYSCLK_Init
//-----------------------------------------------------------------------
//
// 本程序将系统时钟初始化为使用 18.432MHz 晶体作为时钟源。
//
void SYSCLK_Init (void)
{

   int i;                           // 延时计数器
```

```
   OSCXCN = 0x67;                  // 启动使用 18.432MHz 晶体的外部振荡器
   for (i=0; i < 256; i++) ;       // XTLVLD 等待时间(>1ms)
   while (!(OSCXCN & 0x80)) ;      // 等待晶体振荡器稳定
   OSCICN = 0x88;                  // 选择外部振荡器为 SYSCLK 源并允许时钟丢失
                                   // 检测器
}

//-----------------------------------------------------------------------
// PORT_Init
//-----------------------------------------------------------------------
//
// 配置交叉开关和 GPIO 端口
//
void PORT_Init (void)
{
   XBR0    = 0x07;                 // 允许 I2C、SPI 和 UART
   XBR1    = 0x00;
   XBR2    = 0x40;                 // 允许交叉开关和弱上拉
   PRT0CF |= 0xff;                 // 允许 P0 的所有输出为推挽方式;
                                   // 让交叉开关将引脚配置为输入
   PRT1CF |= 0x40;                 // 允许 P1.6 (LED) 为推挽输出
}

//-----------------------------------------------------------------------
// UART_Init
//-----------------------------------------------------------------------
//
// 配置 UART 使用定时器 1 产生波特率<baudrate> 及 8-N-1 的帧格式
//
void UART_Init (void)
{
   SCON   = 0x50;                  // SCON: 方式 1，8 位 UART，允许 RX
   TMOD   = 0x20;                  // TMOD: 定时器 1，方式 2，8 位重装载
   TH1    = -(SYSCLK/BAUDRATE/16); // 根据波特率设置定时器 1 重载值
   TR1    = 1;                     // 启动定时器 1
   CKCON |= 0x10;                  // 定时器 1 使用 sysclk 作为时基
   PCON  |= 0x80;                  // SMOD = 1
   TI     = 1;                     // 指示 TX 准备好
```

```
}

//-----------------------------------------------------------------
// ADC_Init
//-----------------------------------------------------------------
//
// 配置 A/D 转换器使用定时器 3 溢出作为转换启动源，转换完成时产生中断，使用右对齐
// 输出方式。允许 ADC 转换结束中断。保持 ADC 为禁止状态。
//
void ADC_Init (void)
{
   ADC0CN = 0x04;                    // ADC 禁止；正常跟踪方式；ADC 转换由
                                     // 定时器 3 溢出启动；ADC 数据右对齐
                                     REF0CN = 0x07;                    //
                                  允许温度传感器、片内 VREF 和 VREF
                                  // 输出缓冲器
   AMX0SL = 0x0f;                   // 选择温度传感器为 ADC 多路选择器的输出
   ADC0CF = 0x61;                  // ADC 转换时钟 = sysclk/8

   EIE2 |= 0x02;                   // 允许 ADC 中断
}

//-----------------------------------------------------------------
// TIMER3_Init
//-----------------------------------------------------------------
//
// 配置定时器 3 为自动重装载方式,定时间隔由<counts>指定(不产生中断),使用 SYSCLK
// 作为时基
//
void TIMER3_Init (int counts)
{
   TMR3CN = 0x02;                          // 停止定时器 3；清除 TF3；
                                           // 使用 SYSCLK 作为时基
   TMR3RL  = -counts;                      // 初始化重载值
   TMR3    = 0xffff;                       // 立即执行重装载
   EIE2   &= ~0x01;                        // 禁止定时器 3 中断
   TMR3CN |= 0x04;                         // 启动定时器 3
}

//-----------------------------------------------------------------
```

```
// 中断服务程序
//-----------------------------------------------------------------

//-----------------------------------------------------------------
// ADC_ISR
//-----------------------------------------------------------------
//
// ADC 转换中断服务程序
// 我们在此获取 ADC 样本，将其加入到一个总和变量<accumulator>，并将局部抽取计
// 数器<int_dec>减 1。当<int_dec>减到 0 时，我们计算全局变量<result>的新值，
// <result>中保存累加后的 ADC 结果。
//
void ADC_isr (void) interrupt 15
{
   static unsigned int_dec=256;   // 累加/抽取计数器
                                   // 当 int_dec = 0 时我们输出一个新值
   static long accumulator=0L;    // 我们用该变量累加 ADC 样本

   ADCINT = 0;                    // 清除 ADC 转换结束标志

    accumulator += ADC0;              // 读 ADC 值并加到总和变量
   int_dec--;                         // 更新抽取计数器

   if (int_dec == 0) {                // 如果为 0 则执行抽取
      int_dec = 256;                  // 复位计数器
      result = accumulator >> 4;      // 用移位执行除法操作
      accumulator = 0L;               // 累加器清 0
   }
}
```

参考文献

[1] A. Oppenheim and R. Schafer, *Discrete-Time Signal Processing*, New Jersey: Prentice Hall, 1999

[2] J. Lis, *Noise Histogram Analysis*, Cirrus Logic Application Note AN37

J. C. Candy and G. C. Temes, *Oversampling Methods for A/D and D/A Conversion*, IEEE Transactions on Circuits and Systems, June 1987 (Beginning discussion on the effects of oversampling on in-band noise)