

## 使用ADuC8xx MicroConverter®实现RTD接口和线性化

作者: Grayson King和Toru Fukushima

### 简介

铂RTD是最精确的传感器之一, 可用于测量-200°C至+850°C之间的温度, 能够实现±0.02°C或更出色的校准精度。然而, 为获得最高的RTD精度, 需要精确的信号调理、模数转换、线性化和校准。ADI公司的MicroConverter系列产品(<http://www.analog.com/MicroConverter>)包括单芯片上集成一个16位或24位ADC和一个8052 MCU的器件, 再配合信号调理电路, 非常适合RTD传感器。本应用笔记介绍使用ADuC834(或其他类似微转换器)和少量无源元件实现完整RTD传感器接口的方式。

本文引用的软件工具和代码示例, 强烈推荐供使用微转换器实现RTD传感器接口时参考, 这些工具和代码可从以下网址获取: <http://www.analog.com/MicroConverter>。

### 硬件设计

RTD(电阻温度检测器)的阻值是以某一确定的方式随温度变化的函数。在详细了解RTD传递函数的阻值与温度关系(这是非线性的)之前, 假设已经数字化地修正了非线性, 首先重点关注将RTD电阻转换为数字值。一个通常的方法如图1所示。

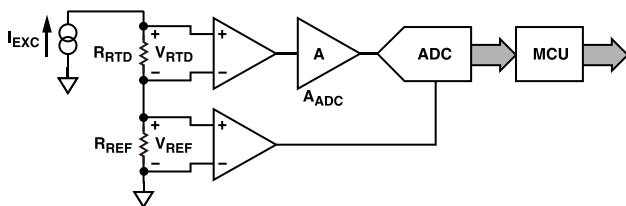


图1. RTD接口硬件配置

这里有单个电流源( $I_{EXC}$ )通过串联方式同时激励RTD( $R_{RTD}$ )和一个精密的基准电阻( $R_{REF}$ ), 从而分别产生ADC输入电压( $V_{RTD}$ )和基准电压( $V_{REF}$ ):

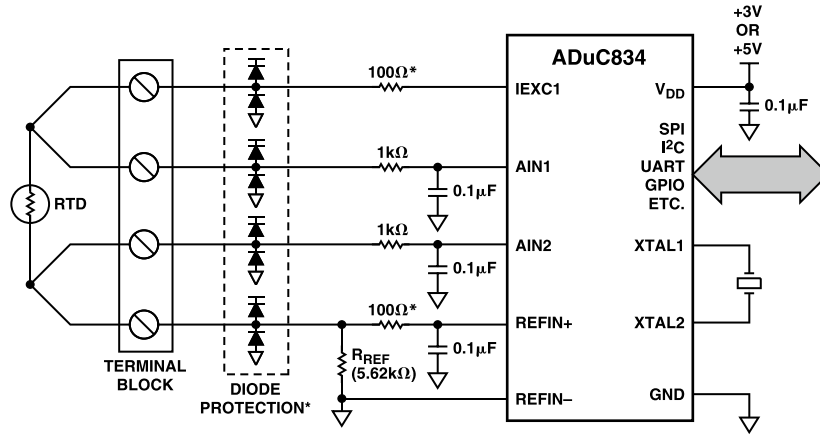
$$\begin{aligned} V_{RTD} &= I_{EXC} \times R_{RTD} \\ V_{REF} &= I_{EXC} \times R_{REF} \end{aligned}$$

ADC归一化的数字输出(零输入=0, 满量程输入=1)就是输入电压与基准电压的比值再乘以增益 $A_{ADC}$ :

$$ADC_{norm} = A_{ADC} \times \frac{V_{RTD}}{V_{REF}} = A_{ADC} \times \frac{I_{EXC} \times R_{RTD}}{I_{EXC} \times R_{REF}} = A_{ADC} \times \frac{R_{RTD}}{R_{REF}}$$

注意 $I_{EXC}$ 是如何在上式中被消掉的, 这意味着, 即使激励电流发生变化或不精确, 该ADC结果始终直接对应于RTD电阻和基准电阻的比值。选择一个高精度、低漂移基准电阻意味着即便使用精度很低的电流源也可以高精度地计算出RTD电阻值。

使用微转换器时应用同一原理, 图2给出了ADuC834与一个4线RTD接口电路的连接。注意, 这是和图1相同的整体拓扑结构, 但所有的有源元件(激励电流源、 $V_{RTD}$ 和 $V_{REF}$ 差分输入级、增益 $A_{ADC}$ 、该ADC本身和一个微控制器)都包括在ADuC834芯片内部, 同时还有许多其他外设, 例如数字通信路径的串行通信端口。注意: 为了实现信号R/C滤波和接线端子过压保护, 电路还要添加一些无源元件。这是一个完整的执行电路, 只需要一个电源和用于数字接口的任意特定外围芯片(例如, RS-232或RS-485线路驱动器/接收器)。



\*DIODE PROTECTION AND 100Ω RESISTORS SERVE ONLY TO PROTECT THE ADuC834 FROM DAMAGE IN THE EVENT OF OVERVOLTAGE CONDITIONS AT THE TERMINAL BLOCK.

图2. 使用ADuC834实现的完整RTD接口电路

## 根据ADC转换结果计算RTD电阻值

根据“硬件设计”部分的描述

$$ADC_{norm} = A_{ADC} \times \frac{R_{RTD}}{R_{REF}}$$

可以写成

$$R_{RTD} = ADC_{norm} \times \frac{R_{REF}}{A_{ADC}} = ADC_{norm} \times scale$$

其中:

$$scale = \frac{R_{REF}}{A_{ADC}}$$

这个标定值(scale)是用在示例代码中的固定比例因子。进一步考虑, 可以将一个固定的偏移值添加到方程中, 因而

$$R_{RTD} = ADC_{norm} \times scale + offset$$

其中偏移(offset)表示一个可用来修正误差的固定偏移量。“偏移”这一词在“校准”部分会有更深入的讨论。大多数情况下, 偏移量设定为零就足够了。请注意, RTD阻值作为ADC转换结果的函数可以用一个方程表示, 此方程只需使用标定值和偏移量就可以决定。

本文档的其余部分则考虑最为通用的铂RTD, 其标称电阻( $R_0$ )在0°C时为100 Ω。同时假设基准电阻值为5.62 kΩ, 该电阻可以很好的匹配这样一个RTD。使用这些元件值和ADuC834, ADC内部增益最高可配置为7.8125就可以满足RTD覆盖其全部的额定温度范围。(记住,  $ADC_{norm}$  根据定义限制在0至1范围内, 它在更高ADC增益的情况下定义了温度范围限制。)7.8125的增益对应于0x4C的ADC0CON值或

320 mV 单极性的范围设置( $A_{ADC} = V_{REF}/span = 2.5 V/320 mV = 7.8125$ )。为了对应于这个增益设置, 标定值设定为719.36 (scale =  $R_{REF}/A_{ADC} = 5.62 k/7.8125 = 719.36$ ), 这也是示例代码中使用的默认标定值。偏移量的默认值为零。

上述 $R_{RTD}$ 方程只是通过软件方法得出的, RTD阻值直接来源于一个给定的ADC转换结果。要确定RTD温度与其阻值变换之间的函数关系, 需要了解RTD传递函数。

## RTD传递函数

铂RTD传递函数是由两个不同的多项式方程描述的: 一个用于温度低于0°C时, 另一个用于温度高于0°C时。这些方程为:

$$R_{RTD}(t) = R_0 [1 + At + Bt^2 + C(t - 100^\circ\text{C})^3] \quad (t \leq 0^\circ\text{C时})$$

$$R_{RTD}(t) = R_0 [1 + At + Bt^2] \quad (t \leq 0^\circ\text{C时})$$

其中:

$t$ 为RTD温度[°C]

$R_{RTD}(t)$ 为RTD阻值, 是RTD温度( $t$ )的函数

$R_0$ 为RTD在0°C时的阻值(通常为100 Ω)

$$A = 3.9083 \times 10^{-3} \text{ } ^\circ\text{C}^{-1}$$

$$B = -5.775 \times 10^{-7} \text{ } ^\circ\text{C}^{-2}$$

$$C = -4.183 \times 10^{-12} \text{ } ^\circ\text{C}^{-4}$$

注意: 符号由 $R_{RTD}$ 变为 $R_{RTD}(t)$ 说明RTD阻值为温度的函数。图3给出了RTD传递函数(阻值为温度的函数)以及过0°C值的线性延伸(为了更形象地比较)。

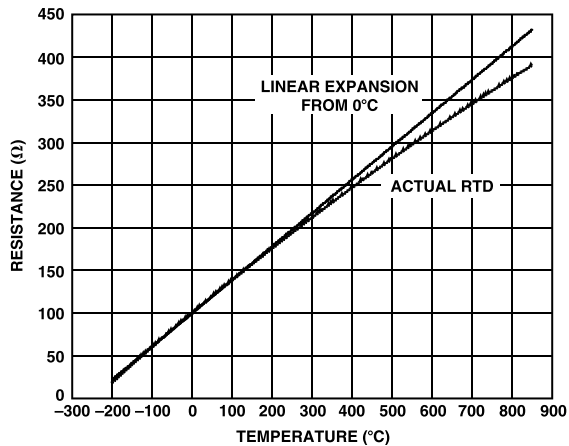


图3. RTD传递函数

前面的方程定义了RTD阻值为温度的函数，即 $R_{RTD}(t)$ 。但是，为了实现一个RTD传感器接口电路，RTD温度必须是确定的，而不是阻值的函数，即 $T_{RTD}(r)$ ，若考虑RTD的非线性传递函数，可能不太易懂。在以下的章节里将探讨一些用于实现该任务的方法。

### 线性化方法

考虑到上述RTD传递函数，有多种方法可以确定作为RTD阻值函数的温度值。本应用笔记考察三种在嵌入式设计中有用的方法，这些方法对基于微转换器的设计尤为适用。表1列出了每种方法的优缺点，同时还总结了每种方法可能最适用的情形。

表1. 三种线性化方法

方法	优点	缺点	总结
直接的数学方法	<ul style="list-style-type: none"> <li>特别精确</li> <li>不需要查找表</li> </ul>	<ul style="list-style-type: none"> <li>需要数学函数库(通常&gt;1 kB)</li> <li>有点慢(10 ms至50 ms*)</li> </ul>	如果其他函数已经要求有数学函数库，则该方法有用
单段线性逼近法	<ul style="list-style-type: none"> <li>非常快(&lt; 1 ms*)</li> <li>需要的代码空间很小</li> <li>在较窄的温度范围内相对精确</li> <li>不需要数学函数库</li> <li>不需要查找表</li> </ul>	<ul style="list-style-type: none"> <li>在较宽的温度范围内不够精确</li> </ul>	适用于代码空间有限且温度范围相对较窄的情形
分段线性逼近法	<ul style="list-style-type: none"> <li>速度快(&lt; 5 ms*)</li> <li>设计人员可权衡控制代码量/精确性</li> <li>可以相当精确</li> <li>不需要数学函数库</li> </ul>	<ul style="list-style-type: none"> <li>相对于单段线性逼近法代码量较多</li> <li>如果其他函数已经要求有数学函数库，则该方法的代码量高于数学方法</li> </ul>	在超过90%的嵌入式设计中可能是这三种方法中最有用的一种

\*这里的执行时间表示ADuC834实证测量值，以12.58 MHz内核时钟速度执行下面的C子程序。

下文将详细介绍这些线性化方法。

### 直接的数学方法

本文前面的部分已经给出了RTD阻值作为其温度函数( $R_{RTD}(t)$ )的详细数学方程。因此我们何不将这些方程变形为RTD温度为其阻值函数( $T_{RTD}(r)$ )的这种表达式呢？这是一个相当简单的任务，因为定义正温度特性的方程仅仅是二次的。解该二次方程得到两个表达式，只需代入几个已知量即可知道哪一个是正确的。结果为下述在 $0^{\circ}\text{C}$ 或者 $0^{\circ}\text{C}$ 以上时RTD温度的方程：

$$T_{RTD}(r) = \frac{-A + \sqrt{A^2 - 4B\left(1 - \frac{r}{R_0}\right)}}{2B}$$

其中 $A$ 、 $B$ 和 $R_0$ 在前文已有定义(“RTD传递函数”部分)， $r$ 为RTD阻值。由于此函数是实时求解的，故化为下面的形式较为有利：

$$T_{RTD}(r) = \frac{Z_1 + \sqrt{Z_2 + Z_3 \times r}}{Z_4}$$

其中：

$$Z_1 = -A = -3.9083 \times 10^{-3}$$

$$Z_2 = A^2 - 4 \times B = 17.58480889 \times 10^{-6}$$

$$Z_3 = \frac{4 \times B}{R_0} = -23.10 \times 10^{-9}$$

$$Z_4 = 2 \times B = -1.155 \times 10^{-6}$$

由于 $Z_1$ 到 $Z_4$ 都是常数且为绝对值，故只需较少的计算量，对于实时计算十分有利。上述关于 $T_{RTD}(r)$ 的方程被称为正函数，这是因为它只是针对大于等于 $0^{\circ}\text{C}$ 时的情况。由于这是一个直接的数学解决方法，它是100%准确的。在解该方程时，8051 C代码的舍入误差为32位浮点型，精度可达到 $+0.0001^{\circ}\text{C}/-0.0005^{\circ}\text{C}$ 左右，对于任何实际用途都可称之为接近100%准确。ADuC832以12.58 MHz的内核时钟速度运行RTDmath.c中的C子程序，该方程的执行时间小于4.2 ms。

上述方程只在温度大于等于 $0^{\circ}\text{C}$ 时有效。定义负温度特性的 $R_{RTD}(t)$ 方程是一个四阶多项式(将第三项展开)，由它解出单一的温度与电阻的函数表达式是不现实的。但是，借助于计算机数学工具可以找出反向传递函数的近似解。运用Mathematica® (<http://www.wolfram.com/products/mathematica>)或其它类似的数学工具软件，可列出以下温度小于等于 $0^{\circ}\text{C}$ 时的最佳拟合多项式表达式：

$$T_{RTD}(r) = -242.02 + 2.2228 \times r + 2.5859 \times 10^{-3} \times r^2 - 4.8260 \times 10^{-6} \times r^3 - 2.8183 \times 10^{-8} \times r^4 + 1.5243 \times 10^{-10} \times r^5$$

$$T_{RTD}(r) = -241.96 + 2.2163 \times r + 2.8541 \times 10^{-3} \times r^2 - 9.9121 \times 10^{-6} \times r^3 - 1.7052 \times 10^{-8} \times r^4$$

$$T_{RTD}(r) = -242.09 + 2.2276 \times r + 2.5178 \times 10^{-3} \times r^2 - 5.8620 \times 10^{-6} \times r^3$$

$$T_{RTD}(r) = -242.97 + 2.2838 \times r + 1.4727 \times 10^{-3} \times r^2$$

这四个方程被称为负函数，这是因为它们每一个都只是针对温度小于等于 $0^{\circ}\text{C}$ 时的情况。最上面的方程(五阶)精度最高，但需要的计算时间最长；最下面的方程(二阶)精度最小，但计算速度最快。表2给出了这些负函数的一些特性，图4给出了每个方程作为温度函数的误差曲线，以及正函数扩展至负温度区域时的误差曲线(作为比较)。由图4我们可以注意到，在 $0^{\circ}\text{C}$ 附近的负温度区域，正函数的误差比二、三、四阶负函数的误差要小。根据这一特性，在较小的负温度区域内使用正函数对RTDmath.c的示例代码将更为有利。实际中选用正函数还是负函数要取决于使用哪个负函数(二阶、三阶、四阶或五阶)，表2给出了各个阈值。大于这个阈值，使用正函数误差较小；小于这个阈值，使用负函数误差较小。表2中方程精度那一栏给出的只是针对低于相应阈值温度的误差。

表2. 最佳拟合多项式方程的特征(负函数)

	最长执行时间*	方程精度*	阈值
五阶	41 ms	+0.0001°C/-0.00005°C	0°C/100 Ω
四阶	31 ms	+0.0022°C/-0.001°C	-8.75°C/96.6 Ω
三阶	21 ms	+0.0053°C/-0.0085°C	-12.5°C/95.1 Ω
二阶	11 ms	+0.075°C/-0.17°C	-70.5°C/72.1 Ω

\*在内核时钟速度为12.58 MHz时，用一个ADuC832芯片运行RTDmath.c中的C语言子程序示例来实际测量执行时间和方程精度。

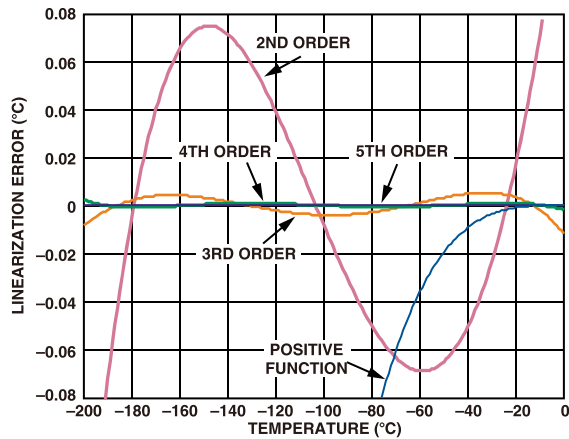


图4. 最佳拟合多项式方程的误差曲线(负函数)

对于线性化，这种直接的数学方法的一个缺点是它需要浮点乘方和开平方根函数，这些函数可以在Keil(<http://www.keil.com>)的C51编译器数学函数库中找到。典型情况下，这些浮点数学函数会单独增加超过1 kB的代码量。使用分段线性逼近的方法可以用更少的代码量获得相似的或更高的精度，该方法在本文稍后部分会有所描述。尽管如此，如果程序中其它操作需要数学函数库中的函数，那么直接的数学方法或许是最好的解决方案，因为这些库函数已经可以使用。

### 单段线性逼近法

在图3中，可以看到在较小的温度范围RTD传递函数近似于一条直线。如果需要检测的温度范围仅仅是全部RTD温度范围的一部分，那么根本不需要对RTD信号进行线性化。在这些情况下，一个在所需温度测量范围的传递函数的最佳拟合线性逼近常常可以获得足够的精度。例如，在工业温度范围-40°C到+85°C时，一个最佳拟合线性逼近的精度为±0.3°C。

一般情况下，作为RTD阻值( $r$ )的一个函数，温度线性方程形式如下

$$T_{LIN}(r) = A \times r + B$$

其中A和B为常量。注意，这里的A和B与“RTD传递函数”部分描述的不同。为A和B选择最优值来最小化误差范围需要一些数学公式，此处并未提及。尽管如此，本应用笔记附带的一个非常简单的软件工具可以自动找出A和B的最优值，来配合您具体的温度范围。这个工具的效果会在本文中验证，但是首先必须确定一个单段线性逼近是否适合具体的设计要求。

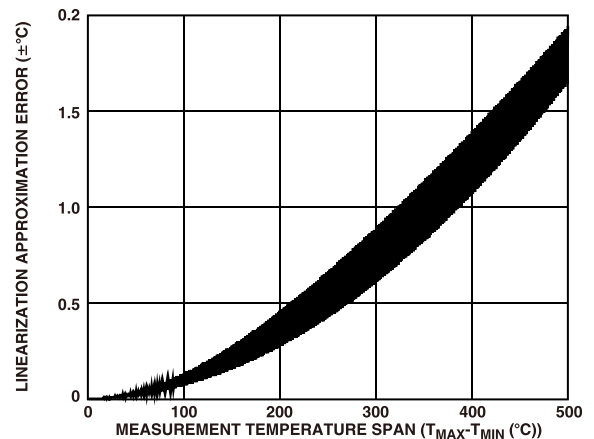


图5. 单段线性逼近误差与测量温度范围

图5给出了500°C以下测量温度范围的全部逼近误差。对于500°C以上的范围，逼近误差会随着温度上升而减小。图5曲线的不精确特性(也就是数据轨迹的宽度范围)源于：即使是相同的温度范围，不同的绝对温度区域误差也是不同的。例如，-200°C至0°C和+600°C至+800°C这两种温度范围，虽然范围都是200°C，但精度却不相同。图5提供了误差的一个较粗略概念，这样可以帮助判断是否考虑使用单段线性逼近。如果确定使用，那么本文稍后介绍的RTD系数生成工具可以帮助确定一个具体温度范围的实际逼近误差，并且生成对于这个温度范围的最优源代码。



## 分段线性逼近法

将线性逼近方法延伸一步，设想用一定数量的直线段可以更好的逼近非线性RTD传递函数。产生这一系列直线段，那么每一个直线段与其相邻直线段之间都会有一个端点，这样看起来就像用一条直线连接一系列的点。可以一次计算出这些点(或系数)来最好的匹配RTD的非线性传递函数，然后将其永久存储在ROM或Flash存储器中。从这些系数表中，MCU可以实现简单的线性插值来确定基于RTD测量电阻的温度值。

为了理解这一过程在实际使用时是如何实现的，首先假设这个系数表已经存在。表中的每一个系数都是传递函数的一个简单的点，由一个电阻和一个温度表示。所以，表格形式为

$$\{r_0, t_0; r_1, t_1; r_2, t_2; \dots; r_N, t_N\}.$$

有了这个表，MCU的实时任务(通过一个给定的电阻 $r$ 来确定温度)就是，首先确定两个最接近所求点的系数( $\{r_m, t_m\}$ 和 $\{r_n, t_n\}$ )，然后在这两个点之间进行线性插值，来解出温度值。对于这个温度范围(只对 $r_m$ 和 $r_n$ 之间的阻值 $r$ 有效)的实际线性插值公式具有以下这种形式

$$T_{SEG}(r) = t_m + (r - r_m) \frac{t_n - t_m}{r_n - r_m}$$

注意上述查找表中的每一个系数都包括两个值，一个是电阻，另一个是温度(实际就是传递函数的 $x$ 和 $y$ 值)。所以，对于 $N$ 个直线段( $N+1$ 个系数)，总共有 $2N+2$ 个值必须存储在存储器中。为了减小查找表的大小，考虑一个包括 $N$ 个直线段的表，每一个直线段都有相同的阻值宽度。这样一个表就可以只存储一系列温度值点，即

$$\{t_0; t_1; t_2; \dots; t_N\}$$

所以，对于一个给定的系数 $\{r_n, t_n\}$ ，可通过下式计算 $r_n$ 的值：

$$r_n = r_0 + n \times r_{SEG}$$

$r_0$ 和 $r_{SEG}$ 是固定值，除系数表之外，它们也存储在ROM中。 $r_0$ 是系数为零 $\{r_0, t_0\}$ 时的阻值， $r_{SEG}$ 是电阻的固定范围，也就

是相邻系数的范围。对于一个给定直线段，线性插值公式变为

$$T_{SEG}(r) = t_i + \left[ r - (r_0 + i \times r_{SEG}) \right] \times \frac{t_{i+1} - t_i}{r_{SEG}}$$

其中 $i$ 表明使用了哪一个直线段(即哪对系数)，并且可以通过 $r$ 计算出来，如下所示：

$$i = \text{trunc} \left( \frac{r - r_0}{r_{SEG}} \right)$$

同样，上述 $T_{SEG}(r)$ 表达式只是 $t_i$ 和 $t_{i+1}$ 这两个系数之间的一个线性插值。要想在实际应用时实现这个过程，MCU必须首先解出 $i$ (通过上述两个方程的后一个)，这样对于 $r$ 来说，系数 $t_i$ 和 $t_{i+1}$ 就是最接近输入值的两个系数。然后，MCU可以简单的求解 $T_{SEG}(r)$ 方程，来确定在给定输入阻值情况下的温度。

这种分段线性逼近法产生的总误差，取决于：1) 分段的数量(系数的个数或查找表的大小)，和2) 总的温度范围。图6给出了 $-200^\circ\text{C}$ 至 $+850^\circ\text{C}$ 这个温度测量范围内的线性逼近误差曲线，作为查找表大小的一个函数(用来优化RTD系数生成工具产生的系数，下面将讨论)。注意，如果测量温度范围减小，相同大小的查找表会给出更好的误差结果，或者一个更小的查找表可以给出同样的误差结果。

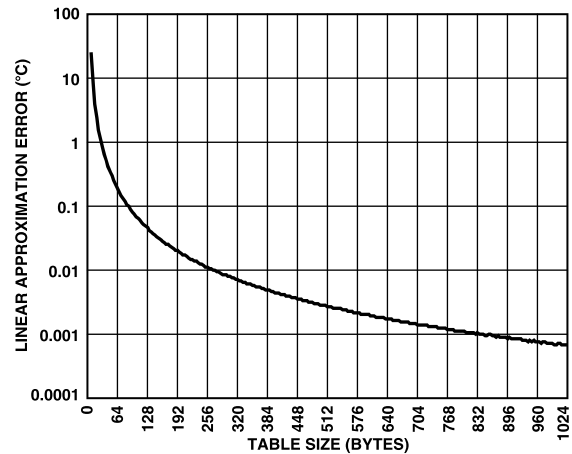


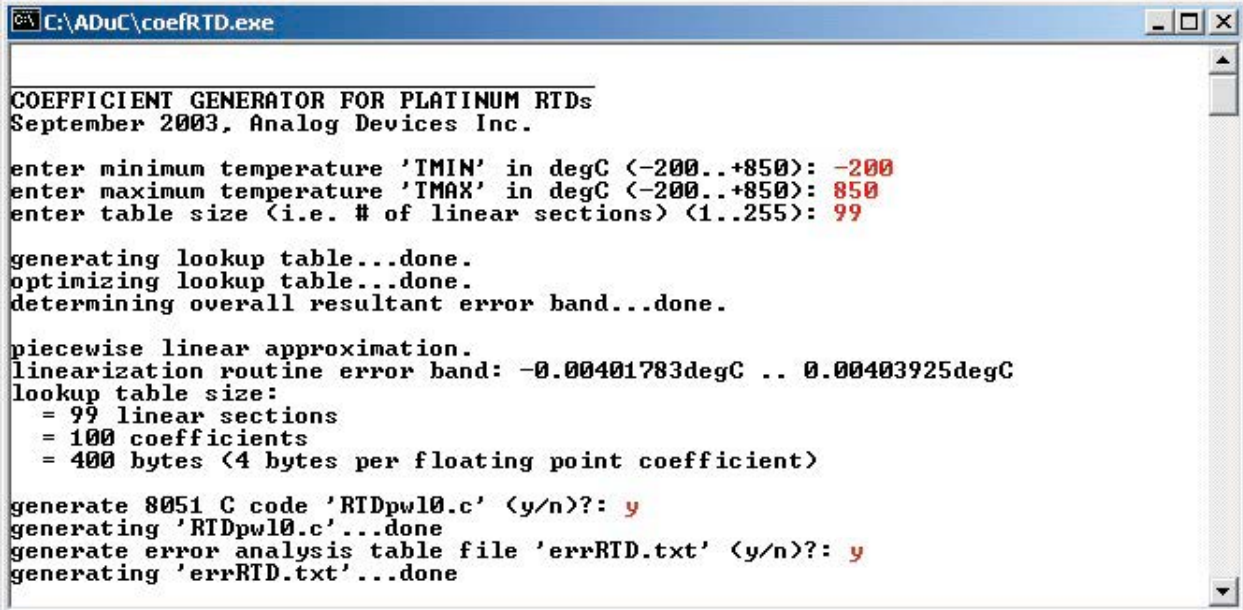
图6. 分段线性逼近误差与查找表大小  
(范围： $-200^\circ\text{C}$ 至 $+850^\circ\text{C}$ )

## RTD系数生成工具

实现分段线性函数，最难的部分当然就是生成查找表。尽管如此，本文附带的RTD系数生成工具(coefRTD.exe)会自动地为铂RTD完成这项工作。这个可以在DOS环境下运行的工具助手(带有基于8051的RTD接口)，就是使用分段线性或单段线性逼近的方法设计的。它可以实现以下任务：

- 可以为一个给定的温度范围和查找表大小生成优化的查找表系数。
- 指明误差范围和查找表大小。
- 用8051 C源代码生成完整的RTD线性化函数(包括查找表)。
- 生成误差表作为已知查找表中温度的一个函数。

图7显示程序运行过程中的一个典型示例画面，其中用户输入内容高亮显示为红色。注意，只需用户输入三个参数( $T_{MIN}$ 、 $T_{MAX}$ 、和 $N_{SEG}$ )，程序就会完成其余所有工作。程序可以生成文件RTDpwl0.c，它是一个完整的C源程序文件(定制一个用户特定查找表)并且可以被包括在一个项目中。在这个项目中， $T_{rtd}()$ 函数可以直接从其它源程序的函数中调用。另外，也可以直接复制和粘贴RTDpwl0.c文件的任意部分到其它源文件中。系数生成工具也可以输出一份误差分析文件(errorRTD.txt)，它是一个表格制式的文本文件，可以导入Microsoft® Excel或任何其它空白表格软件，来检验由线性逼近程序产生的误差。



```

C:\ADuC\coefRTD.exe

COEFFICIENT GENERATOR FOR PLATINUM RTDs
September 2003, Analog Devices Inc.

enter minimum temperature 'TMIN' in degC (-200..+850): -200
enter maximum temperature 'TMAX' in degC (-200..+850): 850
enter table size (i.e. # of linear sections) (1..255): 99

generating lookup table...done.
optimizing lookup table...done.
determining overall resultant error band...done.

piecewise linear approximation.
linearization routine error band: -0.00401783degC .. 0.00403925degC
lookup table size:
  = 99 linear sections
  = 100 coefficients
  = 400 bytes (4 bytes per floating point coefficient)

generate 8051 C code 'RTDpwl0.c' (y/n)? : y
generating 'RTDpwl0.c'...done
generate error analysis table file 'errRTD.txt' (y/n)? : y
generating 'errRTD.txt'...done
  
```

图7. 使用分段线性逼近法生成系数的会话示例(用户输入内容为红色)

```

C:\ADuC\coefRTD.exe
COEFFICIENT GENERATOR FOR PLATINUM RTDs
September 2003, Analog Devices Inc.

enter minimum temperature 'TMIN' in degC (-200..+850): -40
enter maximum temperature 'TMAX' in degC (-200..+850): 85
enter table size (i.e. # of linear sections) (1..255): 1

generating lookup table...done.
optimizing lookup table...done.
determining overall resultant error band...done.

single linear approximation.
linearization routine error band: -0.292923degC .. 0.292929degC
overall equation: t = Ar + B
where:  A = 2.57559degC/ohm
        B = -257.339degC
        r = RTD resistance in ohms
        t = RTD temperature in degC

generate 8051 C code 'RTDlin0.c' (y/n)? : y
generating 'RTDlin0.c'...done
generate error analysis table file 'errRTD.txt' (y/n)? : y
generating 'errRTD.txt'...done

```

图8. 使用单段线性逼近法生成系数的会话示例(用户输入内容为红色)

系数生成程序不仅可为分段线性逼近生成线性化函数，也可为单段线性逼近生成线性化函数。为实现这些，表格大小输入1表示只有单根直线段。程序可进行识别，输出结果适合单段线性化逼近法，而不是分段线性化逼近法，如图8所示。

## 校准

如产品数据手册所述，ADuC834有一个内置的函数用于校准ADC端点误差(偏移和增益误差)。然而，如果不是在校准时，而是考虑整个信号链，包括RTD本身，结果就会得到一个较低的总误差，在这种情况下，内置ADC校准没有额外的益处。本应用笔记首先研究了系统校准，随后列出一些内置ADC校准依然有用的情形。

到目前为止，一直都假设RTD本身是理想的。但是，实际的RTD不是理想的。就像现实世界里的其他事物，RTD与制造商数据手册所描述的指标会有误差。幸运的是，这些误差大都可以很容易地用软件校准。本文讨论的校准函数可用于单点或两点校准，此函数可与之前介绍的任意一种线性化方法配合使用。

若要了解单点校准的工作原理，请参考本文前面讨论的RTD传递函数 $R_{RTD}(t)$ ，注意它大部分是由RTD在0°C时的电

阻值 $R_0$ 定义的。对于最常见的RTD， $R_0$ 一般为100 Ω。但在一个RTD传感器中最主要的误差源就是 $R_0$ 的值，因为不同器件之间差异较大。另外， $R_{RTD}(t)$ 表达式内传递函数的剩余部分需要乘以 $R_0$ ，误差正是由于 $R_0$ 容差是纯粹的乘法所导致的。所以，在将 $R_{RTD}$ 作为ADC转换结果归一化的一个函数时，可以通过调整上述表达式中的标度因子来修正误差。

$$R_{RTD} = ADC_{norm} \times scale + offset$$

特殊情况下，如果RTD测量一个比较精确的已知温度，一次ADC转换已经完成，校准的标定值可由下式计算：

$$scale = \frac{R_{cal}}{ADC_{cal}}$$

其中 $ADC_{cal}$ 是ADC转换的实际标准结果， $R_{cal}$ 是RTD温度理想的(期望的)阻值。可以使用 $R_{RTD}(t)$ 方程手动计算 $R_{cal}$ 。通过这种方法(所谓的单点校准)可以得到一个校准标定值，通过它可以补偿RTD的 $R_0$ 容差，同时还有基准电阻的初始容差。从这里再进一步，可以执行两点校准，它不仅可以补偿这些尺度误差，也可以补偿任何可能存在的偏移误差。这样做不仅需要调整标定值，也需要调整偏移值。现在假设已经实现了单点校准，RTD可以得到另一个非常精确的已知温度，ADC转换又进行了一次。标定值方程(也就是 $R_{RTD}/ADC_{norm}$ 函数的斜率)为



$$scale = \frac{R_{cal} - R_{prevcal}}{ADC_{cal} - ADC_{prevcal}}$$

其中：

$R_{prevcal}$  和  $ADC_{prevcal}$  分别是前一校准点的电阻和ADC转换结果。

对于当前校准点， $R_{cal}$  和  $ADC_{cal}$  是一样的。

注意，这只是采用直线上两点计算  $R_{RTD}$  和  $ADC_{norm}$  传递函数曲线斜率的一种方法。这里只要考虑偏移值，因为标定值现在是已知的，偏移值可以用单点计算。下面偏移值的表达式通过解前面  $R_{RTD}$  表达式的偏移，然后分别用  $R_{prevcal}$  和  $ADC_{prevcal}$  替代  $R_{RTD}$  和  $ADC_{norm}$  得到。

$$offset = R_{prevcal} - ADC_{prevcal} \times scale$$

注意，如果  $R_{prevcal}$  和  $ADC_{prevcal}$  全为零(就是没有前校准点)，那么标定值的表达式就同单点校准一样，只有在单点校准时偏移值表达式为零。因而，单点校准或两点校准可以使用同样的函数(示例代码中的 Cal())。

如果使用示例代码，按照下列步骤完成两点校准：

1. 选择两个温度用于校准，确保温度点充分分开(理想情况下至少有四分之一的整个测量跨度)以避免测量温度末端附近误差累积。
2. 使用RTD测量第一个温度点，等待以保证显示结果建立新值，然后按下终端(或终端仿真器)上的任何一个键，显示用户输入/输出菜单。
3. 按照菜单提示校准到已知温度点，然后在提示下输入温度。
4. 重复第2、3步校准第二个温度点。

注意：对于单点校准，请直接忽略第4步。

尽管这样校准有很多好处，但是在一些应用中，由于需要考虑许多系统因素，因此这种方法不太实用。如果不能实现上述校准过程，那么就需要考虑用一个系统ADC校准来代替(ADuC834数据手册中有所描述)。如果要这样做，可以简单的将RTD短路(0Ω)，并且进行一个系统零标度校准，然后用一个719.36Ω的高精度电阻替代RTD，并且进行一个满量程校准。这是对于内部ADC误差和  $R_{REF}$  电阻初始容差的补偿，但是并未考虑RTD本身的任何误差。

注意，ADuC832(以及所有其他微转换器产品)的一个额外的好处就是，它内部有一个片内非易失性Flash存储器，这个存储器可以用来存储已校准标度以及偏移量。这样，每次系统上电启动时，芯片可以调出存储的校准值，而不需要每次都进行校准。示例代码RTD834.c就出于这种目的利用了该特性。

## 误差分析

在进行数据采集设计时会有很多误差源(ADC线性度、输入放大器噪声、电阻约翰逊噪声、放大器温漂以及电阻温漂等)。对于一项设计而言，确定主要误差源是比较困难的。幸运的是，ADuC834将所有的有源级都集成到一个单独的完全工厂确定的器件中，这使得误差分析变得较为简单。但是在设计中仍然需要考虑非线性传感器元件带来的误差。因此，针对本应用笔记至此已讨论过的特定硬件和软件配置，本文给出了几个最为显著的误差因素。

首先，对于一个具体的RTD，如果没有进行系统校准(采用前文讨论的单点或两点校准方法)，那么RTD本身几乎必然是绝对误差的最显著来源。这个误差取决于所选RTD的具体型号，在RTD制造商的数据手册中，该误差应该很好地量化。本应用笔记则重点讨论除RTD自身以外的误差源。

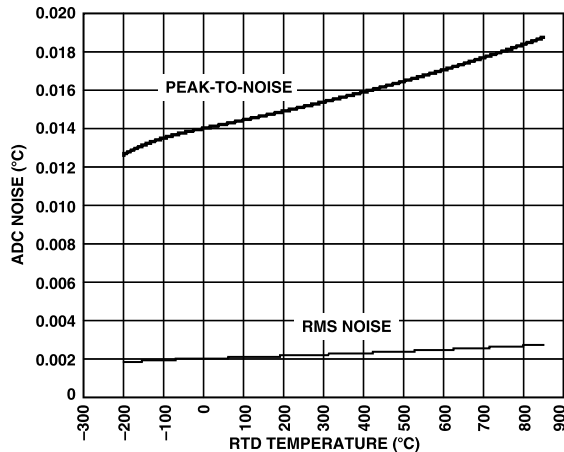


图9. ADC噪声与RTD温度

一类要研究的误差就是噪声。在这种设计中三个主要的噪声源需要考虑：电阻约翰逊噪声、放大器/ADC输入电压噪声和放大器/ADC输入电流噪声。这些噪声以均方根方式累积，从而当一个噪声源只是稍微比其他噪声源大时，较小的作用源是可以忽略的。在这种特殊情况下，主要的噪声源就发生在放大器/ADC输入电压噪声。具体而言，采用所讨论的增益设置时，ADuC834输入电压噪声规格为 $0.37\ \mu\text{V rms}$ ，或大约 $2.44\ \mu\text{V p-p}$ 。由于电阻-温度传递函数的非线性，将输入电压噪声转换为输出温度噪声或许不够直观。作为RTD温度的一个函数，温度噪声的结果是多种多样的。结果如图9所示。注意：即使在最高的RTD温度(也就是最坏噪声)峰峰噪声总是低于 $0.019^\circ\text{C}$ ；它甚至好于较低的测量温度。记住：作为RTD温度函数的噪声变化不是ADC本身的函数，而是如前所述，在数字域使用的非线性 $T_{\text{RTD}}(r)$ 传递函数的直接结果。

另外需要考虑的误差源是温漂：特定ADC偏移、增益温漂和基准电阻温漂。这是DC误差的变化(偏移和增益误差)，可以作为ADC芯片或基准电阻温度变化的一个函数。这取

决于RTD调理电路的环境温度，而不是实际测量的RTD温度。简单来说，此处这两个不同的温度分别指环境温度和RTD温度。另外，由于非线性 $T_{\text{RTD}}(r)$ 传递函数，温漂值(也就是环境温度灵敏度)随着RTD温度函数变化。图10显示了最终结果，但可能需要几点解释。图10的x轴表示RTD温度。y轴表示环境温度每 $^\circ\text{C}$ 变化测量误差温漂的度数。例如，如果RTD温度固定在 $100^\circ\text{C}$ ， $V_{\text{REF}}$ 漂移(使用 $5\ \text{ppm}/^\circ\text{C}$ 基准电阻)约为 $\pm 0.01^\circ\text{C}/^\circ\text{C}$ 。因而，如果环境温度变化，例如 $50^\circ\text{C}$ ，读到的测量温度可能变化 $\pm 0.5^\circ\text{C}$ (忽略其他温漂的影响)。

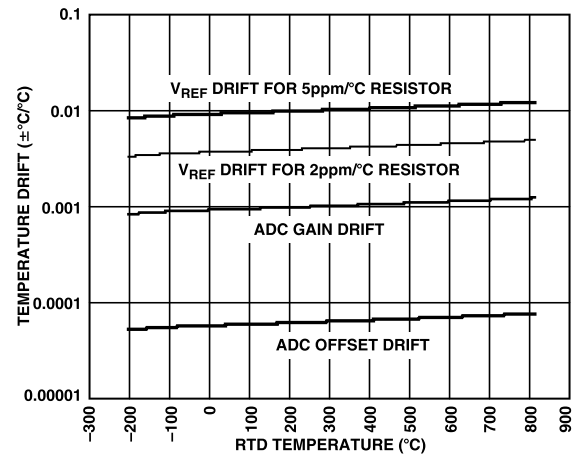


图10. 温漂与RTD温度的关系

不难看出，在工业环境下，环境温度经常在 $-40^\circ\text{C}$ 至 $+85^\circ\text{C}$ 或更宽的范围变化，温漂就成了一个主要的误差源。可以直接使用ADuC834的片内温度传感器测量片内温度(它紧随环境温度变化)，然后使用测量的片内/环境温度补偿温漂误差。在制造时这需要一个额外的温度循环步骤，将环境温度设定为两个固定的值，然后在各环境温度下获取零标度和满量程ADC读数。在此之后，软件可完成其余工作，在内部温度传感器精度范围内来补偿温漂误差，以及补偿外部基准电阻和ADuC834之间的温度梯度误差。本应用笔记没有进一步研究温漂补偿技术，然而请注意片内资源的存在使得仅仅改变软件就可以实现该选项。

RTD自热效应是另外一个需要考虑的误差源。简言之，电流通过RTD时会引起功耗，这会使RTD温度上升。幸运的是，RTD可以使用只有400  $\mu\text{A}$ 的激励， $R_0$ 为100  $\Omega$ 时RTD总的功耗不超过16  $\mu\text{W}$ 。这么小的功耗所引起的自热量是变化的，取决于RTD所使用的特定型号，但是一般情况下导致的自热效应是可以忽略的。如果在特定应用中无法忽略，只需在软件控制下将激励电流降至200  $\mu\text{A}$ ，从而将RTD功耗降低为原来的四分之一。

其他误差源几乎是可以忽略的。直流端点误差(偏移和增益误差)可以使用上述校准方法充分校正。电阻约翰逊噪声远低于ADC输入电压噪声。值得考虑的唯一其他噪声源是ADC INL(积分非线性或相对精度)。ADuC834数据手册中对于最差情形下INL的具体规定是满量程的15 ppm，这会导致折合到输出端的INL误差大约是输出噪声峰峰值的两倍，如图9所示，实际INL非常接近满量程的2 ppm，明显低于本底噪声。这通常是ADI的保守规格。

## 软件和源代码

本文引用的所有软件和源代码都包含在本文随附的一个zip文件里，可从第1页提到的地址下载。zip文件的内容如下：

- coefRTD.exe – 系数生成工具的可执行文件。

- coefRTD.cpp – 系数生成工具源代码。
- RTDmath.c – 使用直接数学线性化方法的线性化子程序。
- RTDpwl0.c – 使用分段线性逼近法的线性化子程序。(可以使用coefRTD.exe程序生成该代码的定制版本。)
- RTDlin0.c – 使用单段线性逼近法的线性化子程序。(可以使用coefRTD.exe程序生成该代码的定制版本。)
- RTD834.c – ADuC834或ADuC836完整RTD接口程序的示例。(利用上述线性化方法中的任意一种。)
- RTD834.hex – RTD834.c和RTDpwl0.c的完整编译版本，可下载并在ADuC834或ADuC836上运行。
- RTD845.c – ADuC845、ADuC847或ADuC848完整RTD接口程序的示例。(利用上述线性化方法中的任意一种。)
- RTD845.hex – ADuC845.c和RTDpwl0.c的完整编译版本，可下载并在ADuC845、ADuC847或ADuC848上运行。
- ReadMe.txt – 提供版本信息、介绍各文件功能的文本文件。

使用上述源代码的完整项目必须包含主程序(RTD834.c、RTD845.c或自编程序)和线性化子程序文件(RTDmath.c、RTDpwl0.c、RTDlin0.c或通过coefRTD.exe工具生成的定制源文件)。各个C源文件的注解中提供了许多详细信息。

如果系统符合Philips公司定义的I<sup>2</sup>C标准规范，则用户在购买ADI公司或其下属机构拥有Philips公司许可的I<sup>2</sup>C器件时，可以获得Philips公司I<sup>2</sup>C专利权之下的许可，以便在I<sup>2</sup>C系统中使用这些器件。

