Application Note

如何读取和解释数字温度传感器输出数据

Ren Schackmann

摘要

数字温度传感器现在已成为行业标准，因为它们具有高精度并且兼容各种数字接口，例如 I2C、SPI、UART、1-Wire®、PWM 和新兴的 I3C MIPI。这些器件即插即用，无需信号调节。

在核心部分，数字温度传感器由偏置或带隙基准、集成或远程温度检测晶体管以及集成模数转换器 (ADC) 组成。请注意，温度传感器中的 ADC 具有不同的分辨率。例如，12 位 ADC 输出通常具有 0.0625°C 的 LSB。在 ADC 处理传感器数据后，原始输出通过数字接口发送，并且必须转换为温度值。这些输出通常使用二进制补码有符号定点表示法，这涉及在两个位位置之间放置一个隐含的二进制点。这种格式在各种不同微控制器和处理器中保持广泛的兼容性，即使是那些不支持浮点的微处理器和处理器也是如此。

本应用手册概述了使用定点数学的算法实现。我们使用“Q 格式”（也称为 *Q 表示法*或 *Q 点*）概念来实现定点表示，以描述和区分温度传感器的典型输出编码。QM.n 标记约定代表了不同的温度传感器输出格式和编码。本文在核心定量层面上对这些概念进行了解释，然后使用 C 代码片段、JavaScript、Python 和 Microsoft® Excel 进行了演示。

内容

表格清单

商标

1 引言

TI 温度传感器的数据编码中采用了二进制补码和 Q 格式。二进制补码是一种对负数进行编码的方法。可用范围（1 位）的一半用于表示数字是否为负数。Q 格式是一种对有理数进行编码的方法。通常会保留四位或更多位来表示 1 和 0 之间的小数值。

1.1 二进制补码

二进制补码是计算机中对有符号整数进行编码的一种常见方法，TI 温度传感器中也采用了该方法。许多编程语言都可以在本地存储和处理二进制补码数据。以二进制补码形式对负数进行编码或解码的理论如下：

1. 获取二进制值并对所有位取反（例如，将 1 变为 0，将 0 变为 1）
2. 将取反的数字加 1，记得将溢出位向左传递

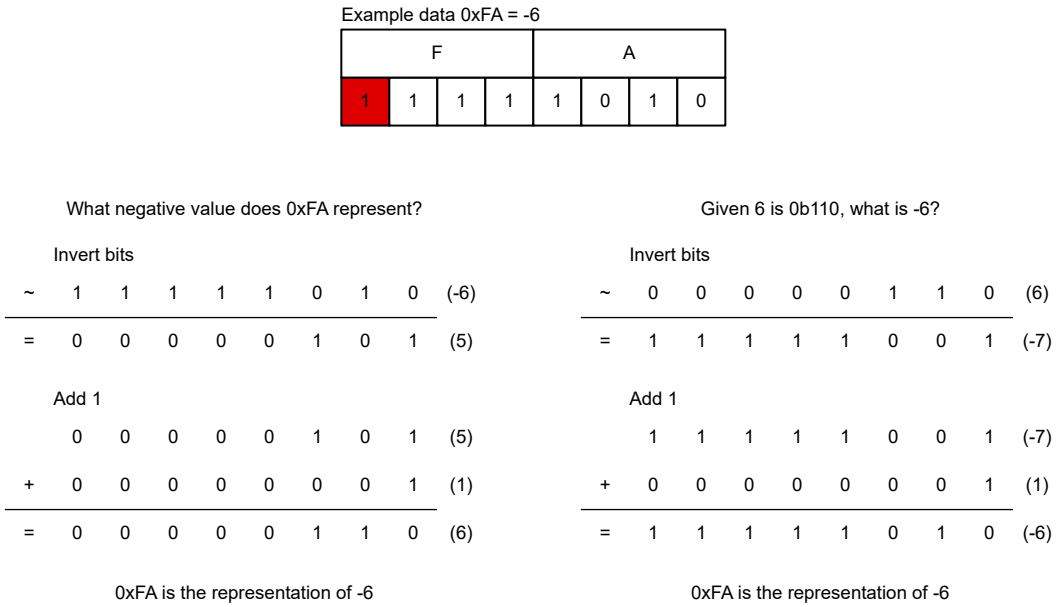

图 1-1. 应用二进制补码的过程

1.1.1 二进制补码特征

表 1-1. 3 位二进制补码表

二进制	有符号值	无符号值
0b100	-4	4
0b101	-3	5
0b110	-2	6
0b111	-1	7
0b000	0	0
0b001	1	1
0b010	2	2
0b011	3	3

此表列出了 3 位二进制补码数的整个取值范围。低位计数使我们能够轻松查看所有可能的值并观察二进制补码编码中的共同特征。

以下是关于二进制补码的一些要记住的事实：

- 最高有效位表示符号。
- 可以表示的**最大数**是 0 后跟全部 1 的二进制数，即此处所示的 0b011。
- 可以表示的**最大绝对值**是 1 后跟全部 0（此处为 0b100），但这**始终是负数**。

- **在二进制中，全 1 始终等于 -1。**
- 将 -1 加上 +1 会导致溢出到 0，这是预期的数学结果。

在下面简化的表格中，我们可以看到相同的特征适用于 8 位二进制补码编码。请注意，“有符号值”列描述了 8 位 C 数据类型 int8_t，“无符号值”列描述了 8 位 C 数据类型 uint8_t。

表 1-2. 8 位二进制补码表

二进制	十六进制	有符号值	无符号值
0b10000000	0x80	-128	128
0b10000001	0x81	-127	129
...	...	...	...
0b11111101	0xFD	-3	253
0b11111110	0xFE	-2	254
0b11111111	0xFF	-1	255
0b00000000	0x00	0	0
0b00000001	0x01	1	1
0b00000010	0x02	2	2
...	...	...	...
0b01111110	0x7E	126	126
0b01111111	0x7F	127	127

1.2 Q 格式

Q 格式是一种对有理数进行编码的方法。通常会保留四位或更多位来表示 1 和 0 之间的小数值。以 Q 格式存储的有理数数据可以高效地处理和存储，而无需进行浮点运算，该运算在微控制器代码中有时是被禁止的。在本文档中，Q 后面的数字是指小数位的位数。如表中所示，其他资料指出了除小数位以外的整数位数，其中 *m* 是包括符号在内的整数位数，*n* 是小数位数。所有参考资料都一致认同一种简便的 *Qn* 表示法，即在 Q 后面只列出 *n* 个小数位，这与本文档是一致的。

表 1-3. Q 表示法变体

资料来源	Q 格式	Q 示例	示例详细信息
本文档	Qn	Q4	共 12 位 8 个整数位，含符号 (7 个整数位，无符号) 4 个小数位
变体 1	Qm.n	Q8.4	
变体 2 (TI)	Q(m-1).n	Q7.4	
变体 3 (ARM)	Qn.m	Q4.8	

Q 格式也可以称为定点数据格式。定点数据具有预先配置的分辨率，而浮点数据具有可变的分辨率。下表展示了所选 Q 格式的分辨率与位权重之间的关系。虽然可以使用更高的 Q 格式，但目前温度传感器中未采用相关格式。

表 1-4. Q 格式比例因子

Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
1	0.5	0.25	0.125	0.0625	0.03125	0.015625	0.0078125
1	1/2	1/4	1/8	1/16	1/32	1/64	1/128
2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}

例如，如果已知整数数据采用 Q4 格式，则可以通过将数据乘以 0.0625、1/16 或 2^{-4} 来转换为有理数 Q 值，因为这些值是相等的。

1.3 常见温度数据格式

大多数数字温度传感器，尤其是具有 I2C 接口的传感器，都采用 12 位 Q4 格式。原始 LM75 传感器采用 Q1 格式并提供 9 位分辨率。LM75 传感器只能通过单个 Q 位以半度为增量报告温度。LM75 的后继产品提供可配置的

9/10/11/12 位分辨率。启用后，这些额外的位将作为 Q 位，并分别提供 Q1/Q2/Q3/Q4 格式。尽管具有额外的位，但这些格式仍然 100% 软件兼容。这种兼容性源于小数点的位置固定不变，如图 1-2 所示；在位数缺失时，输出数据不会在寄存器内移位。

这种格式的一个方便之处在于，如果不需要有理数分辨率，可以极大地简化温度输出的计算。结果的高 8 位表示整数温度，无需执行额外的计算步骤。请参阅下图中的示例，其中高位的值为 32，温度为 32.5625°C。

需要注意的是，这种格式无法表示超过 128°C 的温度，而现代传感器的额定工作温度高达 150°C。

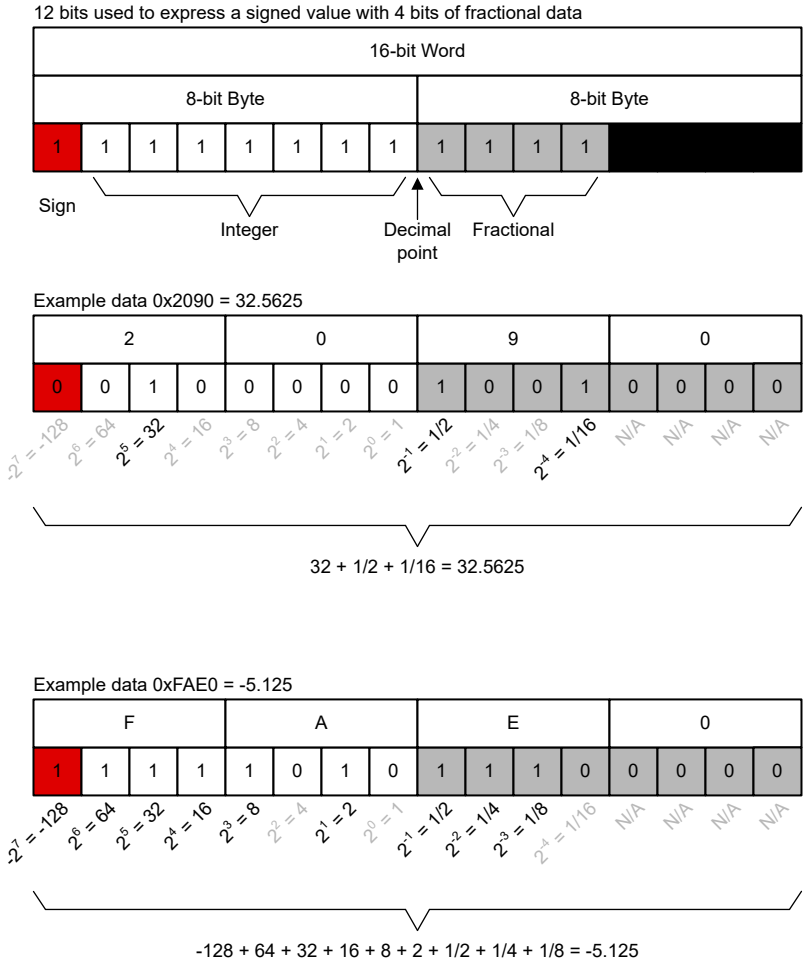

图 1-2. 12 位 Q4 格式

表 1-5. 12 位 Q4 示例数据

温度	数字输出	
	二进制	十六进制
127.9375°C	0111 1111 1111 0000	7FF0
125°C	0111 1101 0000 0000	7D00
25°C	0001 1001 0000 0000	1900
0.0625°C	0000 0000 0001 0000	0010
0°C	0000 0000 0000 0000	0000
-0.00625°C	1111 1111 1111 0000	FFF0
-25°C	1110 0111 0000 0000	E700
-40°C	1101 1000 0000 0000	D800

1.4 高精度温度数据格式

TMP117 等现代传感器以 Q7 格式提供完整的 16 位分辨率。通过将小数点向右移动一位（相对于常见格式），此格式能够表示 +255 到 -256 的范围，但器件的规格并未指定在极端情况下的使用。低位字节中没有未使用的位，因此从 Q7 格式获得了 0.0078125°C 的分辨率。

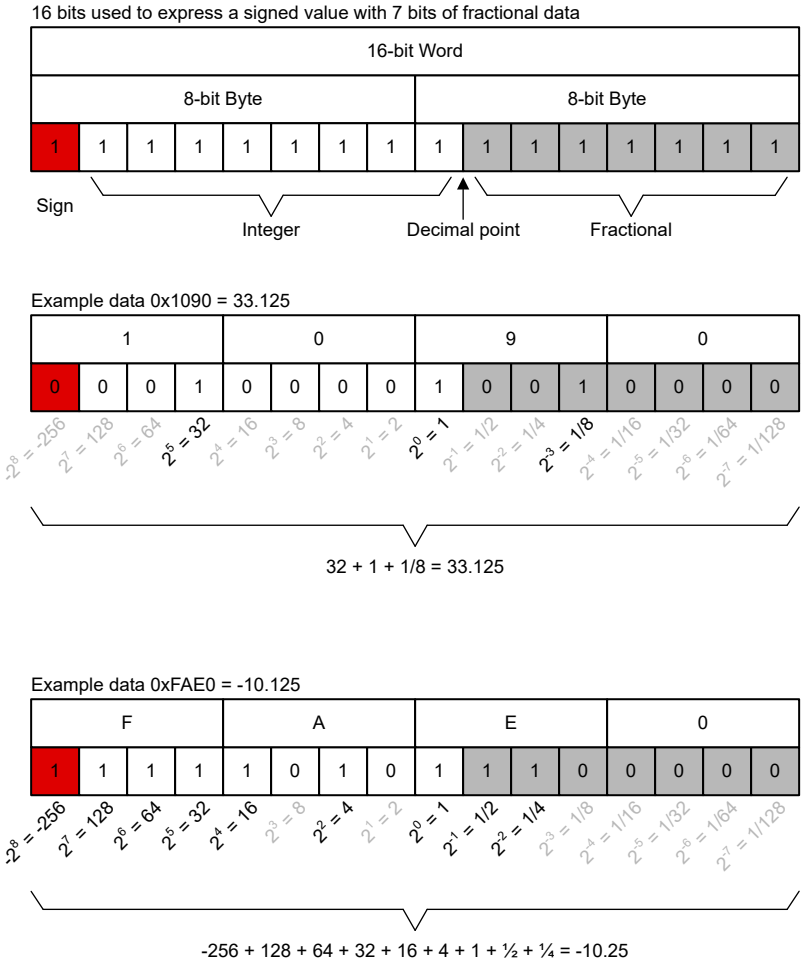

图 1-3. 16 位 Q7 格式

表 1-6. 16 位 Q7 示例数据

温度	数字输出	
	二进制	十六进制
+125°C	0011 1110 1000 0000	3E80
+25°C	0000 1100 1000 0000	0C80
+0.0078125°C	0000 0000 0000 0001	0001
0°C	0000 0000 0000 0000	0000
-0.0078125°C	1111 1111 1111 1111	FFFF
-25°C	1111 0011 1000 0000	F380
-40°C	1110 1100 0000 0000	EC00

2 代码示例

本节提供了适用于现有 TI 产品的现成 C 代码示例。有关编程的详细信息，另请参阅节 3。

本节按位数和 Q 格式将数字温度传感器分成了几组。每个部分都包含一个表格，其中包括位数、Q 格式、分辨率、温度范围、第一个字节温度，以及 25˚C 时的数字输出。**第一个字节整数 C** 行是一个“是或否”描述符，这意味着温度在第一个字节（前 8 位）中表示为整数，因此如果不需要额外的分辨率，则可以丢弃第二个字节。

2.1 16 位（采用 Q7 表示法）

传感器：TMP114、TMP116、TMP117

2.1.1 属性

表 2-1. 16 位 Q7 编码参数

参数	值
位	16
Q	7
分辨率	0.0078125
范围 (+)	255.9921875
范围 (-)	-256
第一个字节整数 C	否
25˚C	0xC80

表 2-2. 16 位 Q7 位值

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
符号	128	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625	0.03125	0.015625	0.0078125
-256	128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	1/32	1/64	1/128
-2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}

2.1.2 C 代码

```
/* 16-bit format will have 0 bits discarded by right shift
   q7 is 0.007812 resolution
   the following bytes represent 24.5C */
uint8_t byte1 = 0xC;
uint8_t byte2 = 0x40;
float f = ((int8_t) byte1 << 8 | byte2) * 0.0078125f;
int mC = ((int8_t) byte1 << 8 | byte2) * 1000 >> 7;
int C = ((int8_t) byte1 << 8 | byte2) >> 7;
```

2.2 12 位（采用 Q4 表示法）

传感器：TMP102、TMP112、TMP1075、TMP75、TMP75B、TMP75C、LM75、LM75B、TMP175、TMP275、TMP108、TMP144、TMP100、TMP101、TMP400、TMP421、TMP422、TMP423、TMP461

这些器件具有 12 位并采用 Q4 表示法。具有这些特性的其他器件包括 LM74 以及任何以 75 结尾的传感器。

这些产品中有多款可配置为 9、10、11 或 12 位输出。每个设置的编码和解码是相同的，无论分辨率配置如何，都可以将 12 位 Q4 解码应用于输出。这是因为未使用的位不会对结果产生任何影响。

2.2.1 属性

表 2-3. 12 位 Q4 编码参数

参数	值
位	12
Q	4
分辨率	0.0625
范围 (+)	127.9375
范围 (-)	-128

表 2-3. 12 位 Q4 编码参数 （续）

参数	值
第一个字节整数 C	是
25˚C	0x1900

表 2-4. 12 位 Q4 位值

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
符号	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625	-	-	-	-
-128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	-	-	-	-
-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	-	-	-	-

2.2.2 C 代码

```
/* 12-bit format will have 4 bits discarded by right shift
   q4 is 0.062500 resolution
   the following bytes represent 24.5C */
uint8_t byte1 = 0x18;
uint8_t byte2 = 0x80;
float f = (((int8_t) byte1 << 8 | byte2) >> 4) * 0.0625f;
int mC = (((int8_t) byte1 << 8 | byte2) >> 4) * 1000 >> 4;
int C = (int8_t) byte1;
```

2.3 13 位 (采用 Q4 表示法) (EM=1)

传感器：TMP102、TMP112、TMP144

这是 TMP102、TMP112 和 TMP144 器件的一种特殊情况。扩展模式位 (EM = 1) 使得这些器件为 13 位而不是 12 位。表示法仍然采用 Q4。

2.3.1 属性

表 2-5. 13 位 Q4 编码参数

参数	值
位	13
Q	4
分辨率	0.0625
范围 (+)	255.9375
范围 (-)	-256
第一个字节整数 C	否
25˚C	0xC80

表 2-6. 13 位 Q4 位值

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
符号	128	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625	-	-	-
-256	128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	-	-	-
-2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	-	-	-

2.3.2 C 代码

```
/* 13-bit format will have 3 bits discarded by right shift
   q4 is 0.062500 resolution
   the following bytes represent 24.5C */
uint8_t byte1 = 0xC;
uint8_t byte2 = 0x40;
float f = (((int8_t) byte1 << 8 | byte2) >> 3) * 0.0625f;
int mC = (((int8_t) byte1 << 8 | byte2) >> 3) * 1000 >> 4;
int C = (((int8_t) byte1 << 8 | byte2) >> 3) >> 4;
```

2.4 13 位 (采用 Q4 表示法)

传感器：TMP468、TMP464、TMP121、TMP122、TMP123、TMP124

2.4.1 属性

表 2-7. 13 位 Q4 编码参数

参数	值
位	13
Q	4
分辨率	0.0625
范围 (+)	255.9375
范围 (-)	-256
第一个字节整数 C	否
25˚C	0xC80

表 2-8. 13 位 Q4 位值

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
符号	128	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625	-	-	-
-256	128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	-	-	-
-2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	-	-	-

2.4.2 C 代码

```
/* 13-bit format will have 3 bits discarded by right shift
   q4 is 0.062500 resolution
   the following bytes represent 24.5C */
uint8_t byte1 = 0xC;
uint8_t byte2 = 0x40;
float f = (((int8_t) byte1 << 8 | byte2) >> 3) * 0.0625f;
int mC = (((int8_t) byte1 << 8 | byte2) >> 3) * 1000 >> 4;
int C = (((int8_t) byte1 << 8 | byte2) >> 3) >> 4;
```

2.5 14 位 (采用 Q6 表示法)

传感器：TMP107

2.5.1 属性

表 2-9. 14 位 Q6 编码参数

参数	值
位	14
Q	6
分辨率	0.015625
范围 (+)	127.984375
范围 (-)	-128
第一个字节整数 C	是
25˚C	0x1900

表 2-10. 14 位 Q6 位值

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
符号	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625	0.03125	0.015625	-	-
-128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	1/32	1/64	-	-
-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	-	-

2.5.2 C 代码

```
/* 14-bit format will have 2 bits discarded by right shift
   q6 is 0.015625 resolution
   the following bytes represent 24.5C */
uint8_t byte1 = 0x18;
uint8_t byte2 = 0x80;
float f = (((int8_t) byte1 << 8 | byte2) >> 2) * 0.015625f;
int mC = (((int8_t) byte1 << 8 | byte2) >> 2) * 1000 >> 6;
int C = (int8_t) byte1;
```

2.6 TMP182x 格式

传感器：TMP1826、TMP1827

这些器件具有一个称为“精密格式”的 16 位 Q7 模式。默认情况下，这些器件使用 12 位传统格式。在本文档中，12 位传统格式实际上是 16 位 Q4。

2.6.1 属性

表 2-11. TMP182x 编码参数

参数	值	值
格式	精密	传统
位	16	16 (12 有效位)
Q	7	4
分辨率	0.0078125	0.0625
范围 (+)	255.9921875	127.9375
范围 (-)	-256	-128
第一个字节整数 C	否	否
25˚C	0xC80	0x190

表 2-12. TMP182x 精密格式位值

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
符号	128	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625	0.03125	0.015625	0.0078125
-256	128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	1/32	1/64	1/128
-2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}

表 2-13. TMP182x 传统格式位值

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
符号	符号	符号	符号	符号	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625
-2048	1024	512	256	128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16
-2^{11}	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}

2.6.2 C 代码

```
/* 16-bit format will have 0 bits discarded by right shift
   q7 is 0.007812 resolution
   the following bytes represent 24.5C */
uint8_t byte1 = 0xC;
uint8_t byte2 = 0x40;
float f = ((int8_t) byte1 << 8 | byte2) * 0.0078125f;
int mC = ((int8_t) byte1 << 8 | byte2) * 1000 >> 7;
int C = ((int8_t) byte1 << 8 | byte2) >> 7;
```

2.7 14 位 (采用 Q5 表示法)

传感器：TMP126、TMP127、LM73、LM95071

2.7.1 属性

表 2-14. 14 位 Q5 编码参数

参数	值
位	14
Q	5
分辨率	0.03125
范围 (+)	255.96875
范围 (-)	-256
第一个字节整数 C	否
25˚C	0xC80

表 2-15. 14 位 Q5 位值

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
符号	128	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625	0.03125	-	-
-256	128	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	1/32	-	-
-2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	-	-

2.7.2 C 代码

```
/* 14-bit format will have 2 bits discarded by right shift
   q5 is 0.031250 resolution
   the following bytes represent 24.5C */
uint8_t byte1 = 0xC;
uint8_t byte2 = 0x40;
float f = (((int8_t) byte1 << 8 | byte2) >> 2) * 0.03125f;
int mC = (((int8_t) byte1 << 8 | byte2) >> 2) * 1000 >> 5;
int C = (((int8_t) byte1 << 8 | byte2) >> 2) >> 5;
```

2.8 8 位 (无 Q 表示法)

传感器：TMP103、TMP104、TMP4718 (本地)

2.8.1 属性

表 2-16. 8 位 Q0 参数

参数	值
位	8
Q	0
分辨率	1
范围 (+)	127
范围 (-)	-128
第一个字节整数 C	是
25˚C	0x19

表 2-17. 8 位 Q0 位值

7	6	5	4	3	2	1	0
符号	64	32	16	8	4	2	1
-128	64	32	16	8	4	2	1
-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

2.8.2 C 代码

```
/* 8-bit format will not have byte2
   q0 is 1.000000 resolution
   the following byte represents 24C */
uint8_t byte1 = 0x18;
int C = (int8_t) byte1;
```

2.9 11 位 (采用 Q3 表示法)

传感器：TMP4718 (远程)

2.9.1 属性

表 2-18. 11 位 Q3 参数

参数	值
位	11
Q	3
分辨率	0.125
范围 (+)	127.875
范围 (-)	-128
第一个字节整数 C	是
25˚C	0x1900

表 2-19. 11 位 Q3 位值

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
符号	64	32	16	8	4	2	1	0.5	0.25	0.125	-	-	-	-	-
-128	64	32	16	8	4	2	1	1/2	1/4	1/8	-	-	-	-	-
-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	-	-	-	-	-

2.9.2 C 代码

```
/* 11-bit format will have 5 bits discarded by right shift
   q3 is 0.125000 resolution
   the following bytes represent 24.5C */
uint8_t byte1 = 0x18;
uint8_t byte2 = 0x80;
float f = (((int8_t) byte1 << 8 | byte2) >> 5) * 0.125f;
int mC = (((int8_t) byte1 << 8 | byte2) >> 5) * 1000 >> 3;
int C = (int8_t) byte1;
```

2.10 不采用二进制补码的器件

以下数字温度传感器不使用二进制补码格式来读取温度值：TMP401、TMP411、TMP431、TMP432、TMP435 和 TMP451。因此，解码不会将它们转换为有符号类型。这些器件表示负温度的方式是启用一个 RANGE 位，该位会将结果增加 64°C。在启用了 RANGE 的情况下，解码必须减去 64，这样原始值 0 变为 -64°C 输出。

2.10.1 属性

表 2-20. 12 位 Q4 参数

参数	值
位	12
Q	4
分辨率	0.0625
范围 (+)	127.9375
范围 (-)	0
第一个字节整数 C	是
25˚C	0x1900

表 2-21. 12 位 Q4 位值

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	64	32	16	8	4	2	1	0.5	0.25	0.125	0.0625	-	-	-	-
-	64	32	16	8	4	2	1	1/2	1/4	1/8	1/16	-	-	-	-
-	2^6	2^5	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	-	-	-	-

2.10.2 C 代码

```
/* 12-bit format will have 4 bits discarded by right shift
   q4 is 0.062500 resolution
   the following bytes represent 24.5C
   there is no cast into signed type */
uint8_t byte1 = 0x18;
uint8_t byte2 = 0x80;
float f = ((byte1 << 8 | byte2) >> 4) * 0.0625f;
int mC = ((byte1 << 8 | byte2) >> 4) * 1000 >> 4;
int C =  byte1;
```

3 其他编程语言

本节详细介绍了使用 C、Microsoft Excel、JavaScript 和 Python 执行温度转换所需的代码。具体过程是解析传入数据，应用二进制补码，丢弃未使用的位，然后应用 Q 格式。节 3.4 中提供了 Excel 和 JavaScript 的完整代码示例。此代码是针对 12 位 Q4 器件编写的，用于其他器件时，尽管原理是相同的，但需要进行调整。

3.1 解析

- 对硬件进行编程时，数据通常作为两个 8 位字节返回。
  - 如果编程语言支持显式类型转换，则最好一开始将这些字节存储为无符号字节，以便更好地控制转换为有符号字节的过程。必须避免的行为是对低位字节进行**符号扩展**。

```
/* C Signed Types */
unsigned char x = 0xFF;
signed char y = 0xFF;
/* x is treated as 255 and y as -1 */

/* C99 fixed width integer types */
uint8_t x = 0xFF;
int8_t y = 0xFF;
/* x is treated as 255 and y as -1 */
```

- 在 C 语言中，**sscanf()** 用于解析字符串 (char*)，其中包括带有 0x 前缀的数字。与之对应的 **printf()** 使用相同的格式字符串。长度修饰符是格式字符串一个不太常用的特性，但此修饰符可以帮助清理小数据类型。下表列出了各种数据类型和格式字符串以及相关的长度修饰符。请注意，scanf 中使用的 %i 可以检测并正确解码 0x 前缀，而 %d 和其他格式说明符则不能。

表 3-1. 带有长度修饰符的 C 格式字符串

位	数据类型	固定宽度类型	格式字符串
8	char	int8_t/uint8_t	%hhi
16	short int	int16_t/uint16_t	%hi
32	int	int32_t/uint32_t	%i

```
/* C Parsing and Outputting Hex */
char *s = "0xFF";
uint8_t x;
sscanf(s, "%hhi", &x);
/* x is 255 */

/* printf without length modifier */
printf("%i, %d, %u, %x\n", x, x, x, x);
/* "-1, -1, 4294967295, ffffffff" is printed due to coercion into 32 bit types and sign-extend */

/* printf with length modifier */

printf("%hhu, 0x%hhX, %#hhX\n", x, x, x);
/* the desired "255, 0xFF, 0XFF" is printed */
```

- 十六进制使用非常方便。Excel 具有 **HEX2DEC** 及相关函数。此函数不支持在前面加上“0x”。HEX2DEC 会将“FFFFFFFFFF”（十个 F）转换为 -1，因此该函数在内部相当于一个 40 位有符号数据类型。这个位数对于存储温度数据来说已经足够了。
- Excel 中的二进制函数（比如 **DEC2BIN**）只能处理 9 位二进制数据。
- **HEX2DEC** 是解码温度数据所需的，而互补函数 **DEC2HEX** 则在编码温度数据时很有用。
- 在下表中，B 列计算的结果为 15 和 0xA。

表 3-2. 用于解析的 Excel 示例

	A	B
1	F	=HEX2DEC(A1)
2	10	=DEC2HEX(A2)

- JavaScript、Python 和 C 都接受带有 0x 前缀的数字常数。

- JavaScript 中的函数 **parseInt()** 用于解析十六进制的“0x”表示法。例如，parseInt() 函数可以正确地“0xA”转换为 10。
- JavaScript 中的 **toString()** 方法用于将数据转换为字符串，如果提供参数 16，则会创建一个十六进制值。

```
/* JavaScript Parsing and Outputting Hex */
let x = 0xA
let y = parseInt("0xA")
let z = (10).toString(16)
let s = "0x" + x.toString(16).toUpperCase().padStart(2,'0')
/* x and y are 10, z is 'a' and s is '0x0A' */
```

- Python 的 **int()** 可以在提供基数参数的情况下转换字符串。
- Python 的 **hex()** 可以将数字转换为十六进制字符串。

```
# Python Parsing and Outputting Hex
x = 0xA
y = int("A",base=16)
z = hex(10)
# x and y are 10, z is '0xa'
```

3.2 二进制补码

- 在支持显式类型转换的编程语言中，在转换为正确的数据类型时会自动处理二进制补码。

```
/* C Type Casting */
unsigned char x = 0xFF;
signed char y = x;
signed int z = (signed char) x;
/* x is 255 but both y and z are -1 */

/* C99 fixed width integer types */
uint8_t x = 0xFF;
int8_t y = x;
int8_t z = (int8_t) x;
/* x is 255 but both y and z are -1 */
```

- 如果不进行类型转换，则需要使用 **IF** 语句来检测并更正应为负数的数字。
  - 对于 n 位的二进制补码数，大于或等于 $2^{(n-1)}$ 的值实际上是负数。换句话说，如果存在 $2^{(n-1)}$ 位，则该数为负数。该位也称为符号位。对于 8 位数字，符号位等于 0x80 或 0b10000000。
  - 要解码负值，请减去 2^n。这是为了将二进制补码负值的顺序颠倒过来。另请参阅表 1-1。
    - 反之也成立：当将负数编码为有符号二进制数据类型时，可以加上 2^n 来获得正确的十六进制/二进制值。
    - 请注意，这种加上或减去 2^n 的方式假定基础数据类型不会溢出。在大多数情况下，这是一个安全的假设，因为如果我们可以将数据类型强制转换为正确的位数，就不需要通过这种方来更正数据的符号。底层数据类型可能是 C 浮点数或类似类型，至少为 32 位。
  - 在 Excel 中，为了提高可读性，可以使用 **IF** 函数和 2^n 语句。Excel 对按位运算的支持相对较弱。减去 2^n 可将数字转换为正确的负值。这些示例适用于 16 位数字。
- 在以下 Excel 示例中，第 1 行展示了对从传感器接收的数据进行解码，而第 2 行展示了对要发送到器件的数据进行编码。

表 3-3. 二进制补码的 Excel 示例

	A	B	C
1	FFFF	=HEX2DEC(A1)	=IF(B1>=2^15,B1-2^16,B1)
2	-1	=IF(A2<0,A2+2^16,A2)	=DEC2HEX(B2)

表 3-4. 二进制补码的 Excel 计算结果

	A	B	C
1	FFFF	65535	-1
2	-1	65535	FFFF

- JavaScript 和 Python 都支持使用十六进制和按位运算符。
  - 0x8000 和 0x10000 相当于 2^{15} 和 2^{16}，无需使用额外的运算符。
  - 按位与比较可用于检查是否存在符号位，而不是进行大于等于的比较。

```
/* JavaScript */
/* decode from 8-bit 2's complement */
function decode(x) {return (x & 0x8000) ? x-0x10000 : x}
let n = decode(0xFFFF)
/* n is -1 */

/* encode to 8-bit 2's complement */
function encode(x) {return (x < 0) ? x+0x10000 : x}
let n = encode(-1).toString(16)
/* n is 'ffff' */
```

```
# Python
# decode from 8-bit 2's complement
def decode(x): return x-0x10000 if (x & 0x8000) else x
n = decode(0xFFFF)
# n is -1

# encode to 8-bit 2's complement
def encode(x): return x+0x10000 if (x & 0x8000) else x
n = hex(encode(-1))
# n is '0xffff'
```

3.3 丢弃未使用的位

- 一些器件在右侧存在未使用的位（最低有效位），必须将其丢弃。
- 在支持位移操作的语言中，丢弃是通过右移操作 (>>) 来实现的。
- 在需要左移或无法避免符号扩展的情况下，可以通过掩码操作来消除不需要的位。掩码操作最好通过按位与来实现。掩码是一个常数，其中需要保留的位设置为逻辑 1。例如，掩码 0xF 等同于 0b1111，并且只保留四个最低有效位。

```
/* right shift */
unsigned char x = 0x23;
unsigned char y = x >> 4;
/* y is 0x02 */

/* mask to discard bits */
unsigned char z = x & 0xF;
/* z is 0x03 */
```

- 在 Excel 中，**乘法**和**除法**运算相当于左移和右移。例如，乘以 2^3 相当于左移 3。右移可以使用 $*2^{-b}$ 或 $/2^b$ 来实现。以这种方式进行右移时，结果必须舍入为整数以丢弃低位。这里使用 INT() 函数来向下舍入。
- BITAND() 可用于掩码操作，但要注意此函数接受十进制输入并提供十进制输出。这可能会让使用 BITAND() 进行掩码操作变得令人困惑。与 HEX2DEC() 类似，BITAND() 至少可以处理 32 位的数据。
- 以下 Excel 示例展示了对 0x23 执行移位和丢弃操作，如其他语言示例中的操作相同。请注意，在此示例中，Excel 在 C 列中显示了十进制结果。我们的结果恰好与其他示例中以十六进制显示的结果相同。第 1 行是移位操作，第 2 行是丢弃操作。

表 3-5. 位丢弃操作的 Excel 示例

	A	B	C
1	23	=HEX2DEC(A1)	=INT(B1*2^-4)
2	23	=HEX2DEC(A2)	=BITAND(B2,HEX2DEC("F"))

表 3-6. 位丢弃操作的 Excel 计算结果

	A	B	C
1	23	35	2
2	23	35	3

- JavaScript 和 Python 支持移位以及按位与。

```
/* JavaScript */
/* right shift */
let x = 0x23
let y = x >> 4
/* y is 0x2 */

/* mask to discard bits */
let z = x & 0xF
/* z is 0x3 */
```

```
# Python
# right shift
x = 0x23
y = x >> 4
# y is 0x2

# mask to discard bits
z = x & 0xF
# z is 0x3
```

3.4 应用 Q 格式

- 该值必须转换为浮点数据类型，然后乘以器件分辨率。
- 如果需要避免使用浮点数据类型，则可以通过右移来丢弃小数位的温度数据。或者，可以使用整数数据类型来存储 mC（毫摄氏度）数据，方法是在右移之前将温度值乘以 1000。

```
/* C Decode 12-bit Q4 */
unsigned char b1 = 0xff;
unsigned char b2 = 0xf0;

/* combine 8 bit bytes into 16 bit word,
 apply signed type cast to upper byte */
int x = (signed char) b1 << 8 | b2;

/* shift to discard unused bits */
int y = x >> 4;

/* Q4 is 2^-4 = 1/16 = 0.0625
 cannot use shift operators on float
 use multiply or divide to create right shift */
float a = y * 0.0625f;

/* discard Q4 bits for a whole number result */
int b = y >> 4;

/* scale by 1000 then shift by Q# to get
 fractional result without float data type */
int c = y * 1000 >> 4; /* milliCelsius */

/* a is -0.0625, b is -1, and c is -63 */
printf("x:%d y:%d a:%f b:%d c:%d\n",x,y,a,b,c);
```

```
/* C Encode 12-bit Q4 */
float in = -0.0625f;

/* Q4 is 2^-4 = 1/16 = 0.0625
 cannot use shift operators on float
 emulate left shift using multiply */
short int r = in * 16;

/* left shift to create unused/discard bits */
short int s = r << 4;

/* s is 0xFFF0 */
printf("r:%d s:%d sx:%hx",r,s,s);
```

- 表 3-7 显示了 12 位 Q4 的完整 Excel 设计。第 1 行展示了对从传感器接收的数据进行解码，而第 2 行展示了对要发送到器件的数据进行编码。

表 3-7. Q 格式的 Excel 示例

	A	B	C	D	E
1	1810	=HEX2DEC(A1)	=IF(B1>=2^15,B1-2^16,B1)	=INT(C1*2^-4)	=D1*0.0625
2	-0.0625	=IF(A2<0,A2+2^8,A2)	=INT(B2*2^4)	=C2*2^4	=DEC2HEX(D2)

表 3-8. Q 格式的 Excel 计算结果

	A	B	C	D	E
1	1810	6160	6160	385	24.0625
2	-0.0625	255.9375	4095	65520	FFF0

以下代码展示了一个针对 12 位 Q4 的完整 JavaScript 和 Python 解码设计，可用于读取温度数据。

```
/* JavaScript */
function decode(x) {return (((x & 0x8000) ? x - 0x10000 : x) >> 4) * 0.0625}
let x = decode(0x1810)
let y = decode(0xFFF0)
/* x is 24.0625 and y is -0.0625 */
```

```
# Python
def decode(x): return ((x-0x10000 if (x & 0x8000) else x) >> 4) * 0.0625
x = decode(0x1810)
y = decode(0xFFF0)
# x is 24.0625
# y is -0.0625
```

以下代码展示了一个针对 12 位 Q4 的温度编码 JavaScript 和 Python 设计，可用于对温度限制或偏移设置进行编码。

```
/* JavaScript */
function encode(x) {return ((x < 0 ? x + 0x100 : x) * 16) << 4}
let x = encode(24.0625).toString(16)
let y = encode(-0.0625).toString(16)
/* x is '1810' and y is 'fff0' */
```

```
# Python
def encode(x): return int((x+0x100 if (x < 0) else x) * 16) << 4
x = hex(encode(24.0625))
y = hex(encode(-0.0625))
# x is '0x1810'
# y is '0xfff0'
```

4 总结

德州仪器 (TI) 提供各种数字温度传感器，这些传感器能够从 ADC 中获取值并将其转换为摄氏度。此过程根据每个器件的位大小和 Q 格式并遵循二进制补码方法来完成。本应用手册中每个部分的代码示例展示了器件执行各种操作的方式，并提供了一个表格，重点介绍了一些关键特性，解释了器件为什么以其特定的方式运行。有关器件应用和布局建议的更多信息，请参阅各个器件的数据表。

5 参考资料

有关本应用手册中引用的器件的更多信息，另请参阅：

- 德州仪器 (TI)，*数字温度传感器* 产品页面

有关本应用手册中提到的计算和表示法，另请参阅以下资源：

- 德州仪器 (TI)，*TMS320C64x DSP 库编程人员参考* 用户指南
- Cornell University, *Two's Complement*, reference site
- Wikipedia, *Fixed-point arithmetic*, article
- Wikipedia, *Q (number format)*, article
- Wikibooks, *Floating Point Unit*, article
- ARM, *RealView Development Suite AXD and armsd Debuggers Guide*, user's guide
- Intel, *Reference Manual for Intel® Integrated Performance Primitives for Microcontrollers*, user's guide

有关用于转换温度数据的具体编程语言，另请参阅以下资源：

- Microsoft®, *HEX2DEC function*, support article
- Mozilla® Corporation, *parseInt()*, JavaScript programming reference
- Wikipedia, *C data types*, article
- Wikipedia, *Operators in C and C++*, article

6 附录：Q 应用源代码

此实用程序可用于对任何 Q 格式编码进行温度转换。此代码专为类 Unix 的 shell 编写和测试。我们建议将源代码另存为 q.c 并使用 cc q.c -o qapp 编译代码。

```
#include <stdio.h>
#include <unistd.h>

void main(int argc, char *argv[])
{
    int c;
    uint8_t bits = 16;
    uint8_t qnum = 7;
    uint8_t byte1;
    uint8_t byte2;
    int16_t data;
    int mode = 0;
    while ((c = getopt(argc, argv, "b:q:hx")) != -1)
        switch (c)
        {
        case 'b':
            sscanf(optarg, "%hhi", &bits);
            break;
        case 'q':
            sscanf(optarg, "%hhi", &qnum);
            break;
        case 'h':
            mode = 1; /* print help */
            break;
        case 'x':
            mode = 2; /* print example code */
        }
    uint8_t shift = 16 - bits;
    float resolution = (float)1 / (1 << qnum);
    /* recommend using a constant for resolution in application code */
    if (mode == 0)
    { /* parse byte(s) and print celsius */
        switch (argc - optind)
        {
        case 2:
            sscanf(argv[optind], "%hhi", &byte1);
            sscanf(argv[optind + 1], "%hhi", &byte2);
            data = (int8_t)byte1 << 8 | byte2;
            printf("input: %d (0x%hhx) and %d (0x%hhX) becomes %d (0x%hX)\n", byte1, byte1, byte2,
byte2, data, data);
            break;
        case 1:
            sscanf(argv[optind], "%hi", &data);
            printf("input: %d (0x%hX) is assumed to be 16-bit\n", data, data);
            break;
        case 0:
            data = 0x7FFF;
            printf("demo: %d (0x%hX)\n", data, data);
            break;
        }
        printf("%d-bit format will have %d bits discarded by right shift\n", bits, shift);
        data >>= shift;
        printf("q%d is %f resolution\n", qnum, resolution);
        float f = data * resolution;
        printf("float: %f C\n", f);
        /* preserve fractional result while using int only */
        int mC = data * 1000 >> qnum;
        printf("int: %d mC\n", mC);
        /* discard fractional result while using int only */
        int C = data >> qnum;
        printf("int: %d C\n", C);
    }
    else if (mode == 1) /* print help */
    {
        printf("qapp [-b bits] [-q qnum] [byte] [byte2]\n\
use -b to specify number of bits\n\
use -q to specify Q format, which describes resolution\n\
byte can be 16 bit data or the first of two 8 bit bytes\n\
byte2 is the second 8 bit byte that will be assembled with byte\n\
\n\
Device settings:\n\
TMP117, TMP114:\n\
```

```
 qapp -b 16 -q 7 0x0C80\n\
TMP102, TMP112, TMP1075:\n\
 qapp -b 12 -q 4 0x1900\n\
TMP102, TMP112 EM=1:\n\
 qapp -b 13 -q 4 0x0C80\n\
TMP468:\n\
 qapp -b 13 -q 4 0x0C80\n\
TMP107:\n\
 qapp -b 14 -q 6 0x1900\n\
TMP126:\n\
 qapp -b 14 -q 5 0x0C80\n\
");
    }
    else if (mode == 2) /* print example code */
    {
        /* use 24.5 to represent nominal, ambient temperature with the first fractional bit set */
        int16_t exdata = 24.5f / resolution;
        exdata <<= shift;
        printf("C Code Examples:\n");
        printf("/* %d-bit format will have %d bits discarded by right shift\n", bits, shift);
        printf("   q%d is %f resolution\n", qnum, resolution);
        printf("   the following bytes represent 24.5C */ \n");
        printf("uint8_t byte1 = 0x%hhX;\n", exdata >> 8);
        printf("uint8_t byte2 = 0x%hhX;\n", exdata);
        if (shift)
            printf("float f = (((int8_t) byte1 << 8 | byte2) >> %d) * %gf;\n", shift, resolution);
        else
            printf("float f = ((int8_t) byte1 << 8 | byte2) * %gf;\n", resolution);
        if (shift)
            printf("int mC = (((int8_t) byte1 << 8 | byte2) >> %d) * 1000 >> %d;\n", shift, qnum);
        else
            printf("int mC = ((int8_t) byte1 << 8 | byte2) * 1000 >> %d;\n", qnum);
        if (shift + qnum == 8)
            printf("int C = (int8_t) byte1;\n");
        else if (shift)
            printf("int C = (((int8_t) byte1 << 8 | byte2) >> %d) >> %d;\n", shift, qnum);
        else
            printf("int C = ((int8_t) byte1 << 8 | byte2) >> %d;\n", qnum);
    }
}
```

7 附录：器件概要表

器件	位	Q	分辨率	范围 (+)	范围 (-)	第一个字节	25C
TMP117	16	7	0.0078125	255.9921875	-256	否	0x0C80
TMP116	16	7	0.0078125	255.9921875	-256	否	0x0C80
TMP114	16	7	0.0078125	255.9921875	-256	否	0x0C80
TMP102	12	4	0.0625	127.9375	-128	是	0x1900
TMP112	12	4	0.0625	127.9375	-128	是	0x1900
TMP1075	12	4	0.0625	127.9375	-128	是	0x1900
TMP75	12	4	0.0625	127.9375	-128	是	0x1900
TMP75B	12	4	0.0625	127.9375	-128	是	0x1900
TMP75C	12	4	0.0625	127.9375	-128	是	0x1900
LM75	12	4	0.0625	127.9375	-128	是	0x1900
LM75B	12	4	0.0625	127.9375	-128	是	0x1900
TMP175	12	4	0.0625	127.9375	-128	是	0x1900
TMP275	12	4	0.0625	127.9375	-128	是	0x1900
TMP108	12	4	0.0625	127.9375	-128	是	0x1900
TMP144	12	4	0.0625	127.9375	-128	是	0x1900
TMP100	12	4	0.0625	127.9375	-128	是	0x1900
TMP101	12	4	0.0625	127.9375	-128	是	0x1900
TMP400	12	4	0.0625	127.9375	-128	是	0x1900
TMP421	12	4	0.0625	127.9375	-128	是	0x1900
TMP422	12	4	0.0625	127.9375	-128	是	0x1900
TMP423	12	4	0.0625	127.9375	-128	是	0x1900
TMP461	12	4	0.0625	127.9375	-128	是	0x1900
TMP102，EM=1	13	4	0.0625	255.9375	-256	否	0x0C80
TMP112，EM=1	13	4	0.0625	255.9375	-256	否	0x0C80
TMP144，EM=1	13	4	0.0625	255.9375	-256	否	0x0C80
TMP468	13	4	0.0625	255.9375	-256	否	0x0C80
TMP464	13	4	0.0625	255.9375	-256	否	0x0C80
TMP121	13	4	0.0625	255.9375	-256	否	0x0C80
TMP122	13	4	0.0625	255.9375	-256	否	0x0C80
TMP123	13	4	0.0625	255.9375	-256	否	0x0C80
TMP124	13	4	0.0625	255.9375	-256	否	0x0C80
TMP107	14	6	0.015625	127.984375	-128	是	0x1900
TMP1826（精密）	16	7	0.0078125	255.992187	-256	否	0x0C80
TMP1827（精密）	16	7	0.0078125	255.9921875	-256	否	0x0C80
TMP1826（传统）	16 (12)	4	0.0625	127.9375	-128	是	0x190
TMP1827（传统）	16 (12)	4	0.0625	127.9375	-128	是	0x190
TMP126	14	5	0.03125	255.96875	-256	否	0x0C80
TMP127	14	5	0.03125	255.96875	-256	否	0x0C80
LM73	14	5	0.03125	255.96875	-256	否	0x0C80
LM95071	14	5	0.03125	255.96875	-256	否	0x0C80

器件	位	Q	分辨率	范围 (+)	范围 (-)	第一个字节	25C
TMP103	8	0	1	127	-128	是	0x19
TMP104	8	0	1	127	-128	是	0x19
TMP4718（本地）	8	0	1	127	-128	是	0x19
TMP4718（远程）	11	3	0.125	127.875	-128	是	0x1900
TMP401	12	4	0.0625	127.9375	0	是	0x1900
TMP411	12	4	0.0625	127.9375	0	是	0x1900
TMP431	12	4	0.0625	127.9375	0	是	0x1900
TMP432	12	4	0.0625	127.9375	0	是	0x1900
TMP435	12	4	0.0625	127.9375	0	是	0x1900
TMP451	12	4	0.0625	127.9375	0	是	0x1900

8 修订历史记录

Changes from Revision * (April 2024) to Revision A (January 2025)	Page
• 更新了整个文档中的表格、图、商标和交叉参考的编号格式	1

重要通知和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的相关应用。 严禁以其他方式对这些资源进行复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265