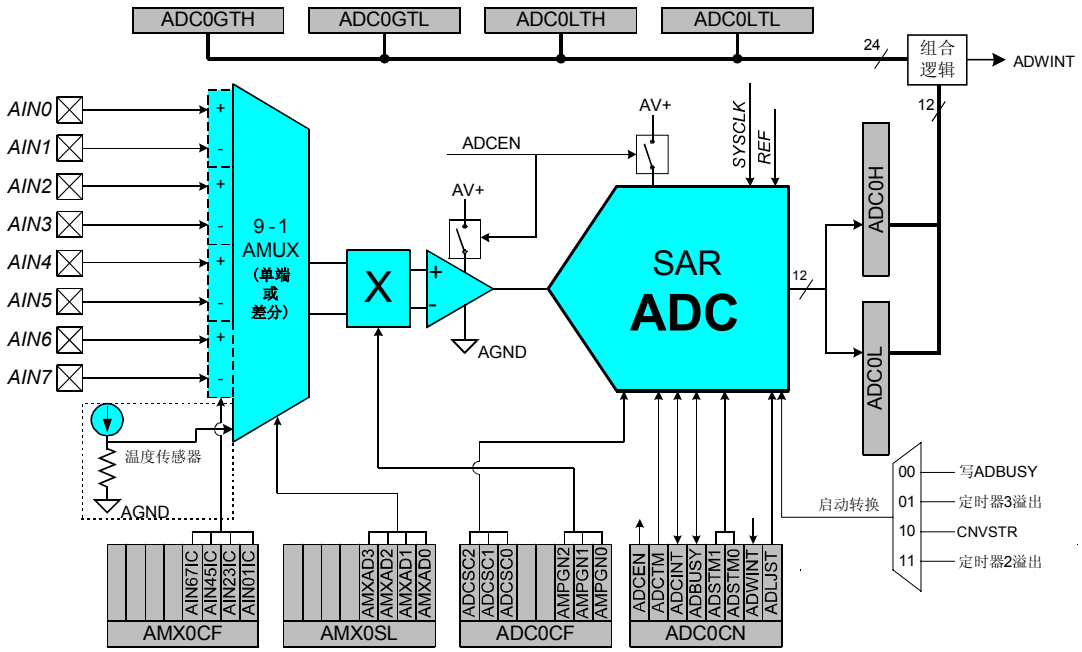
AN003 — 使用片内温度传感器



引言

本应用笔记的目的是介绍如何配置和使用片内温度传感器。本文提供了配置说明和示例代码。

温度传感器产生一个与器件基材温度成正比的电压，该电压作为一个单端输入提供给ADC（模数转换器）的多路开关。当选择温度传感器作为ADC的一个输入并且ADC启动一次转换后，可以经过简单数学运算将ADC的输出结果转换成用度数表示的温度。

温度传感器的应用包括系统环境监视、系统过热测试和基于热电偶的应用中，测量冷端温度。

关键点：

- 温度传感器的分辨率可以通过求均值得到改善。
- 温度传感器测量的是器件的基材温度。如果希望测量环境温度，则必须考虑器件的自热效应。

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

配置说明

为了使用温度传感器，它首先必须被允许。ADC及其相关的偏置电路也必须被允许。ADC可以使用内部电压基准也可以使用外部电压基准。本应用笔记中的例子使用内部电压基准。ADC转换的结果代码可以选择为左对齐或右对齐。本应用笔记中的例子使用左对齐，这样可使代码的权值与ADC的位数无关；也就是说，所用的公式和常数适用于具有9位到16位ADC输出的所有器件。

通过将TEMPE（REF0CN.2）设置为‘1’来允许温度传感器工作。模拟偏置发生器和内部电压基准的允许位也位于REF0CN中（分别为REF0CN.1和REF0CN.0）；所有这些位可以在一次写操作中被允许，例如：

```
mov     REF0CN, #07 h
;允许温度传感器、模拟偏置产生器和电压基准
```

下一步，必须选择温度传感器为ADC的输入，这可以通过写AMX0SL来完成，例如：

```
mov     AMX0SL, #0f h       ; 选择温度传感器作为ADC输入
```

AMX0CF的值为何以及AMUX配置寄存器选择ADC是单端输入还是叉分输入，并不影响温度传感器工作。

下一步，必须正确设置位于ADC0CF中的ADC SAR时钟分频系数。特别是ADC转换时钟的周期至少应为400ns。表1给出所需的最小分频系数与SYSCLK的关系。

表1. ADC时钟分频系数与SYSCLK的关系

SYSCLK频率	ADC分频系数	ADCSC2-0
时钟频率 < 2.5 MHz	1	000
2.5 MHz – 5 MHz	2	001
5 MHz – 10 MHz	4	010
10 MHz – 20 MHz	8*	011
时钟频率 > 20 MHz	16	1xx

*表示复位值

接下来选择ADC的增益。在单端方式下，ADC能够接收的最大直流输入电压等于VREF。如果使用内部电压基准，该值大约为2.4 V。温度传感器所能产生的最大电压值稍小于1V。因此，我们可以安全地将ADC的增益设置为‘2’，以提高我们的温度分辨率。设置ADC增益的配置位在ADC0CF中。我们有：

```
mov     ADC0CF, #61h        ; 设置ADC的时钟 = SYSCLK/8
                            ; 设置ADC增益 = 2
```

其余的ADC配置位位于ADC0CN，这是一个可以位寻址的寄存器。可以选择任何一种有效的转换启动机制：定时器2或定时器3溢出，向ADBUSY写‘1’，或用外部CNVSTR。下面的软件示例使用定时器3溢出作为转换启动源。这里我们采用向ADBUSY写‘1’。

通过写入下面的控制字，我们将ADC配置为低功耗跟踪方式，采用向ADBUSY写‘1’作为转换启动信号，输出数据采用左对齐格式：

```
mov     ADC0CN, #c1h        ; 允许ADC；允许低功耗跟踪方式
                            ; 清除转换完成中断
                            ; 选择ADBUSY作为转换启动源
                            ; 清除窗口比较中断
                            ; 设置输出数据格式为左对齐；
```

至此，我们可以通过向ADBUSY写‘1’来启动一次转换：

```
setb    ADBUSY              ; 启动转换
```

现在我们等待转换完成：

```
jnb     ADCINT, $           ; 等待转换完成
```

一旦转换完成，ADC输出寄存器，即ADC0H和ADC0L中的16位数值包含与器件基材的绝对温度成正比的代码。下面一节告诉我们如何通过这一代码得到温度的摄氏度数值。

结果阐释

温度传感器产生一个与器件基材绝对温度成正比的电压输出。方程1给出这一电压和温度的摄氏度数值之间的关系。

方程1

$$Vtemp = 2.5\ mV/C * Temp + 0.603\ V$$

其中：

Vtemp = 温度传感器的输出电压
Temp = 器件基材的摄氏温度值

温度传感器的传输特性如图1所示。

温度传感器的电压不能直接在器件外面观察到。它出现在ADC多路开关的输入端，允许ADC测量该电压值并产生一个与电压值成正比的输出代码。ADC在左对齐、单端方式下产生的代码与输入电压成正比，如下所示：

方程2

$$CODE = Vin * (Gain / Vref) * 2^{16}$$

其中：

CODE = 左对齐的ADC输出代码
Gain = ADC的增益
Vref = 电压基准的电压值，如果使用内部Vref，则大约为2.4 V。

把方程1代入方程2，我们得到一个与温度的摄氏度数成正比的输出代码：

CODE = (2.5 mV/C * Temp + 0.603 V) * (Gain / Vref)* 2^16

为了求解Temp，我们重写方程，得到：

方程3

Temp = (CODE - K1) * K2

其中：

K1 = 0.603 * (Gain / Vref) * 2^16

K2 = 1 / (2.5 mV / C * (Gain / Vref) * 2^16

对于 Vref = 2.4V 和 Gain = 2：

K1 = 32,932 = 0x80a4

K2 = 480 / 2^16 = 0x01d4 / 2^16

因为K2是一个分数，为便于实现，在所有的计算中都保持其值为480，而在最后除以2^{16}。

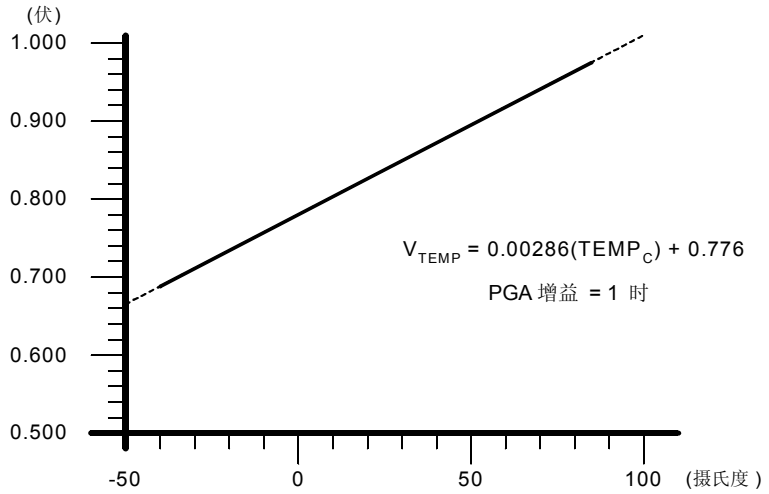


图1．温度传感器传输特性。

实现时的考虑

自热

温度传感器测量的是器件基材的湿度，测量值很可能比环境温度值高几度，这是由于器件功率消耗的结果。为了得到环境湿度，从结果中减去因自热产生的温度增加值。这一温度增加值可以通过计算或测量得到。

有很多因素影响器件的自身发热量。其中最主要的是：电源电压、工作频率、封装的热耗散特性、器件在PCB中的安装方式以及封装外壳周围的空气流通。温度增加值可以通过将器件的功率消耗乘以封装的热耗散常数（通常称为 θ_{JA}）来计算。在用这一常数时定采用标准的PCB安装方式，

所有的引脚都焊到电路板上，封装周围没有空气流通。

对一个工作在11.0592MHz、采用3.3 V电源电压的C8051F000而言，功率消耗大致为35 mW。对于64脚的TQFP封装，其 θ_{JA}值是39.5℃/W。这等价于$39.5*35e^{-3}$的自热值，大约相当于1.4℃。

因自热而导致的温度增加可以用几种方式来测量。一种方法是在器件上电之后立刻启动一次转换，得到一个‘冷’温度值。然后，在工作大约经过1分钟之后再测量一次，得到一个‘热’温度值。这两个测量值的差就是自热产生的影响。

另一种方法是让器件从一个低的SYSCLK频率开始工作，例如32 kHz，进行一次温度测量，然后再让器件工作在标准的SYSCLK频率，例如11.0592 MHz，取两者之差。在较低时钟频率时自热值是可忽略的，因为此时器件的功耗很低。

求均值

为了使温度转换结果中的噪声效应降至最低，一种技术是对数据进行‘过采样’，然后求均值。‘过采样’意味着ADC的采样频率被设置为高于输出字速率。作为一种经验方法，你可以通过每将采样频率增加到四倍而得到一位额外的输出分辨率。

软件示例

例 1

```
;-----------------------------------------------------------------------
; CYGNAL INTEGRATED PRODUCTS, INC.
;
; 文件:     Temp_1.ASM
; 目标MCU: C8051F000，C8051F010
;
; 本程序提供一个通过ADC配置和使用片内温度传感器的例子
; ADC被设置为左对齐方式，所以本程序不用经过修改就可用于10位或12位ADC。
;
; 用内部振荡器作为系统时钟，使用其缺省工作频率（~1.9MHz）。
;
; ADC被设置为左对齐方式，GAIN = 2，使用定时器3溢出作为转换启动源。
; 定时器3被配置为自动重装载方式，每20ms溢出一次。
; ADC转换完成中断处理程序读ADC的值得并将其与保存在ROOMCODE常数中的
; 室内温度（约25℃）期望值进行比较。如果所测得的温度值低于ROOMCODE，
; 则LED 熄灭。如果所测得的温度值高于ROOMCODE，则LED点亮。
;
; 通过改变ROOMCODE的值可以很容易地修改LED的切换点。
;
;-----------------------------------------------------------------------

;-----------------------------------------------------------------------
; 等价定义
;-----------------------------------------------------------------------

$MOD8F000

LED             EQU     P1.6            ; 目标板LED控制 ('1' 时LED亮)

TC_20MS         EQU     38000           ; 1.9MHz 时的定时器滴答数，对应20ms

ROOMCODE        EQU     9800h           ; 25℃所对应的左对齐ADC值
;-----------------------------------------------------------------------
; 变量
;-----------------------------------------------------------------------

BSEG

                org     0h

DSEG 位于 30h

; ADC 数据变量

TEMPCODE:       DS      2                   ; 温度码保持寄存器 (16位)
;--------------------
; 间接地址空间变量

ISEG 位于 80h
```

```
;--------------------
; 堆栈

              org    0e8h              ; 临时设置的堆栈地址

STACK_TOP:       DS     1              ; (rev C 勘误)
                                       ; 符号表中的占位符，表示硬件堆栈起始地址

;-----------------------------------------------------------------------
; 宏定义
;-----------------------------------------------------------------------

;-----------------------------------------------------------------------
; 复位和中断向量表
;-----------------------------------------------------------------------

CSEG
          org    00h
          ljmp   Main

          org    7bh
          ljmp   ADC0EOC_ISR          ; ADC0 转换结束中断
;-----------------------------------------------------------------------
; 主程序代码
;-----------------------------------------------------------------------

          org    0B3h

Main:
          mov    WDTCN, #0deh         ; 禁止看门狗定时器
          mov    WDTCN, #0adh

          mov    SP, #STACK_TOP       ; 初始化堆栈指针

          mov    XBR2, #40h           ; 允许交叉开关和弱上拉

          orl    PRT1CF,#01000000b    ; 设置P1.6（目标板上的LED控制）为弱上拉
          acall  ADC0_Init            ; 初始化ADC和温度传感器
          acall  TIMER3_Init          ; 初始化定时器3

          acall     TIMER3_Start      ; 允许定时器3
          acall     ADC0_Enable       ; 允许ADC

          setb      EA                ; 允许全局中断

          sjmp      $                 ; 原地跳转

;-----------------------------------------------------------------------
; 中断向量
;-----------------------------------------------------------------------
;-----------------------------------------------------------------------
; ADC0EOC_ISR
;
; 该ISR在ADC转换完成后被调用。当该事件发生时，ADC的值被拷贝到保持
```

```
; 变量TEMPCODE，并与25℃对应的代码比较。如果温度值高于25℃，
; 则LED点亮。如果温度值低于25℃，则LED熄灭。没有考虑自热校正。
;
ADC0EOC_ISR:
              push   PSW
              push   acc

              clr    ADCINT                ; 清除ADC中断标志

              mov    TEMPCODE, ADC0L       ; 拷贝ADC结果之低字节到TEMPCODE
              mov    TEMPCODE+1, ADC0H     ; 拷贝ADC结果之高字节到TEMPCODE

       ; 将TEMPCODE与对应于25摄氏度的期望值比较
       ; 如果(TEMPCODE - ROOMDEG) < 0，则熄灭LED，否则点亮。
       ; 计算TEMPCODE - ROOMREG 并存于TEMPCODE (16-bit 减法)

              clr    C
              mov    a, TEMPCODE           ; 减去低字节
              subb   a, #LOW(ROOMCODE)
              mov    TEMPCODE, a           ; 保存新的低字节
              mov    a, TEMPCODE+1         ; 减去高字节（带借位）
              subb   a, #HIGH(ROOMCODE)
              mov    TEMPCODE+1,a          ; 保存新的高字节

              setb   LED                   ; 点亮LED。
              jnc    ADC0EOC_ISR_END       ; 如结果为正则返回，
              clr    LED                   ; 熄灭LED后返回

ADC0EOC_ISR_END:
              pop    acc
              pop    PSW

              reti
;-------------------------------------------------------------------------
; 子程序
;-------------------------------------------------------------------------
;-------------------------------------------------------------------------
; TIMER3_Init
;-------------------------------------------------------------------------
; 该子程序将定时器3配置为：自动重装载方式、在TC_20MS时溢出、用SYSCLK作为计数时钟。
; 返回前停止定时器3并禁止其中断。
;
TIMER3_Init:
              mov    TMR3CN, #02h            ; 停止3，清除TF3，
                                             ; 用SYSCLK作为时基
              mov    TMR3RLH, #HIGH(-TC_20MS); 初始化重载值
              mov    TMR3RLL, #LOW(-TC_20MS)
              mov    TMR3H, #0ffh            ; 立即重装载
              mov    TMR3L, #0ffh
              anl    EIE2, #NOT(00000001b)      ; 禁止定时器3中断

              ret
```

```
;-----------------------------------------------------------------------
; TIMER3_Start
;-----------------------------------------------------------------------
; 该子程序启动定时器3
;
TIMER3_Start:
              orl TMR3CN, #00000100b      ; 置位 TR3
              ret

;-----------------------------------------------------------------------
; ADC0_Init
;-----------------------------------------------------------------------
; 该子程序初始化 ADC 为左对齐方式，用于监测片内温度传感器，增益为2，禁止 ADC
;
ADC0_Init:
              clr    ADCEN                ; 禁止 ADC
              mov    REF0CN, #07h         ; 允许温度传感器，偏置发生器
                                          ; 和输出缓冲器
              mov    AMX0SL, #0fh         ; 选择温度传感器为 ADC 输入
              mov    ADC0CF, #01100001b   ; ADC 转换时钟 = SYSCLK/8
                                          ; 对于 18.432MHz，GAIN = 2x
              mov    ADC0CN, #01000101b   ; 禁止ADC，低功耗跟踪方式，
                                          ; 由定时器3溢出启动ADC转换，
                                          ; 数据左对齐
              ret

;-----------------------------------------------------------------------
; ADC0_Enable
;-----------------------------------------------------------------------
; 该子程序允许ADC和ADC中断
;
ADC0_Enable:
              setb   ADCEN                ; 允许 ADC
              orl    EIE2, #00000010b     ; 允许 ADC 转换结束中断
              ret

;-----------------------------------------------------------------------
; 文件结束。

END
```

例 2

```
//-----------------------------------------------------------------------
// CYGNAL INTEGRATED PRODUCTS, INC.
//
// 文件:       Temp_2.c
// 目标MCU:    C8051F000，C8051F010
//
// 本程序将C8051Fxxx 的基底温度由硬件UART输出，波特率为115.2kbps，8个数据位，
// 无奇偶校验，一个停止位。假定在XTAL1和XTAL2之间连接一个11.0592MHz的晶体。
//
// 工具链：KEIL C51
//
// Make.bat 为:
// C:\Keil\C51\BIN\C51.EXE temp_2.c CD OE DB SB
// C:\Keil\C51\BIN\BL51.EXE temp_2.obj
//

//-----------------------------------------------------------------------
// 编译命令
//-----------------------------------------------------------------------

// 编译器命令行选项

//-----------------------------------------------------------------------
// 包含文件
//-----------------------------------------------------------------------

#include <c8051f000.h>                  // SFR 声明
#include <stdio.h>

//-----------------------------------------------------------------------
// 全局常数
//-----------------------------------------------------------------------

#define ON              1
#define OFF             0
#define XTLVLD_BIT  0x80                // OSCXCN.7 晶体振荡器有效标志
#define TC_20MS     18432               // 定时器在 11.0592MHz/12时对应于
                                        // 20ms的嘀答数
#define LED             P1.6            // LED='1' 表示点亮



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

void sysclk_init (void);
void xbar_init (void);
void uart_init (void);
void ADC_init (void);
void ADC_enable (void);
```

```
void Timer3_init (void);
void ADC_isr (void);

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

long   temperature;                 // 以百分之一度表示的温度值
unsigned  idata temp[16];           // 温度采样值的循环缓冲区
int temp_ptr;                       // 指向 temp[]的指针
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）

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

void main (void) {

    sysclk_init ();                 // 初始化振荡器
    xbar_init ();                   // 初始化交叉开关和GPIO
    uart_init ();                   // 初始化 UART
    Timer3_init ();                 // 初始化定时器3为 20ms 溢出
    ADC_init ();                    // 初始化 ADC 的输入为温度传感器
    ADC_enable ();                  // 允许 ADC转换结束中断

    temperature = 0L;
    temp_ptr = 0;

    EA = 1;                         // 允许中断

    while (1) {
           // 将以百分之一度表示的温度值转换成两个十进制数，整数部分和小数部分
           temp_int = temperature / 100;
           temp_frac = temperature - (temp_int * 100);
           printf ("Temperature is '%d.%02d'\n", temp_int, temp_frac);
    }
}
// 将器件配置为使用外部 CMOS 时钟。
void sysclk_init (void)
{
    WDTCN = 0xde;                   // 禁止看门狗定时器
    WDTCN = 0xad;

    // 启动外部振荡器
    OSCXCN = 0x65;                  // 对于11.0592MHz的晶体，允许SYSCLK/1

    // 等待外部振荡器起动
    while ( (OSCXCN & XTLVLD_BIT) == 0 )
    {
    }

    OSCICN = 0x88;                  // 选择外部振荡器作为系统时钟
                                    // 禁止内部振荡器
```

```
}

// 配置交叉开关和 GPIO 端口
void xbar_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口的所有输出为弱上拉;
                                        //   让交叉开关将这些引脚配置为输入
    PRT1CF |= 0x40;                     // 允许 P1.6 (LED) 为弱上拉输出
}

// 配置 UART 使用定时器1，115.2k的波特率， 8-N-1 帧格式，使用 11.0592MHz sysclk
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为重装载
    TH1    = -6;                        // TH1: 重载值，115.2kbps @ 11.0592MHz
    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
}
```

```
// 配置 A/D 转换器使用定时器3溢出为转换启动源，并在转换完成时产生中断
void ADC_init (void)
{
    ADCEN  = 0;                          // 禁止 ADC
    REF0CN = 0x07;                       // 允许温度传感器，片内偏置发生器
                                         //   和偏置输出缓冲器
    AMX0SL = 0x0f;                       // 选择温度传感器为ADC多路开关的输出
    ADC0CF = 0x61;                       // ADC 转换时钟 = sysclk/8，
                                         //   对11.0592MHz，GAIN = 2x
    ADC0CN = 0x45;                       // 禁止ADC，低功耗跟踪方式，
                                         //   ADC 由定时器3溢出启动，数据左对齐
}

// 允许 ADC；定时器3溢出启动转换；允许ADC中断
void ADC_enable (void)
{
    ADCEN = 1;                           //允许 ADC
    EIE2 |= 0x02;                        //允许ADC中断（正常优先级）
}

// 配置定时器3为自动重装载方式，定时间隔为 20ms（不产生中断）
void Timer3_init (void)
{
    TMR3CN = 0;                          // 停止定时器3，清除TF3，使用SYSCLK/12
                                         //   作为时基
    TMR3RLH = 0xff & ((-TC_20MS) >> 8);// 初始化重载值
    TMR3RLL = 0xff & (-TC_20MS);
    TMR3H   = 0xff;                      // 立即重装载
    TMR3L   = 0xff;
    EIE2   &= ~0x01;                     // 禁止定时器3中断
    TMR3CN |= 0x04;                      // 启动定时器3
}

// ADC 转换结束 ISR
//  在此我们读 ADC 采样值，将其放入到16个采样值的缓冲区内，求所有采样值的平均值
//  并将其转换为以百分之一摄氏度为单位的温度值
void ADC_isr (void) interrupt 15
{
    int i;                               // 循环计数器
    long temp_temp;                      // 临时温度值

    ADCINT = 0;                          // 清除ADC转换结束中断标志

    temp_temp = (ADC0H << 8) | ADC0L; // 装入16位ADC结果

    temp[temp_ptr] = temp_temp;          // 加到缓冲区中
    temp_ptr++;
    temp_ptr = temp_ptr % 16;            // 回绕缓冲区指针

    temp_temp = 0L;
```

```
    for (i=0; i<16; i++) {
        temp_temp = temp_temp + temp[i]; // 累加采样值，以便求均值
    }

    temp_temp = temp_temp >> 4;       // 总和/16，求得平均值

    // 至此，temp_temp 中为16个采样值得平均值。
    //现在我们需要将其转换为以百分之一摄氏度表示的温度值。
    //
    // 在左对齐方式，ADC 产生的代码（CODE）与输入电压成正比：
    // CODE = Vin * Gain / Vref * 2^16
    //
    // 温度传感器产生的电压（Vtemp）与以摄氏度表示的绝对温度（Temp）成正比：
    // Vtemp = 2.5mV/C * Temp + 0.603V
    //
    // 结合这两个方程，我们可以将输出码（左对齐）与输入温度的关系表示为：
    // CODE = (2.5mV/C * Temp + 0.603V) * Gain / Vref * 2^16
    //
    // 解上面的方程求Temp：
    // CODE = 2.5mV/C * Temp * Gain/Vref * 2^16 + 0.603V * Gain/Vref * 2^16
    // Temp = (CODE-(0.603V * Gain/Vref * 2^16))/(2.5mV/C * Gain/Vref * 2^16)
    //
    // 可以表示为：
    // Temp = (CODE - K1) * K2
    //
    // 其中 K1 = 0.603 * Gain/Vref * 2^16
    //      K2 = 1/(2.5mV/C * Gain/Vref * 2^16)
    //
    // 对于 Vref = 2.4V 和 Gain = 2,
    // K1 = 32,932 = 0x80a4
    // K2 = 480 / 2^16 - 我们在所有计算中保持该值为480，在最后除以2^16。480=0x01d6

    temp_temp = temp_temp - 0x80a4;   // 将偏差校正到 0度， 0V
    temp_temp = temp_temp * 0x01d6;   // 2.5mV/摄氏度
    temp_temp = temp_temp * 100;      // 显示结果，以百分之一摄氏度表示
    temperature = temp_temp >> 16;    // 除以 2^16
}
```