





# AN003 - Using the On-Chip Temperature Sensor

## Key Points

- The resolution of the temperature sensor can be improved by averaging.
- The temp sensor measures the die temperature of the device. If a measurement of ambient temperature is desired, then the effects of device self-heating must be taken into consideration.

## Configuration Description

In order to use the temp sensor, it must first be enabled. The ADC and its associated bias circuitry must also be enabled. The ADC can use either the internal or an external voltage reference. The examples in this note use the internal voltage reference. The resulting ADC code is selectable to be either left-justified or right-justified. The examples in this note use left-justification, which makes the code weights independent of the number of bits (12 or 10) in the ADC.

The temp sensor is enabled by setting TEMPE (REF0CN.2) to a '1'. The enable bits for the analog bias generator and internal voltage reference are also located in REF0CN (REF0CN.1 and REF0CN.0 respectively); all of these can be enabled in a single write, as follows:

```

; enable temp sensor, analog bias
; generator, and voltage reference
mov REF0CN, #07h

```

Next, the temp sensor must be selected as the input to the ADC, which is accomplished by a write to AMX0SL as follows:

```

; select temp sensor as ADC input
mov AMX0SL, #0fh

```

The value of AMX0CF, the AMUX Configuration Register that selects whether an ADC

input is single-ended or differential, does not affect the temp sensor.

Next, the ADC SAR clock divider, located in ADC0CF, must be properly set. Specifically, the ADC conversion clock must have a period that is at least 500 ns. Table 1 below shows the minimum required clock divider value vs. SYSCLK.

**Table 1. SAR Clock vs. SYSCLK**

SYSCLK freq	ADCSC2-0
CLK < 2.0 MHz	000
2.0 MHz - 4.0 MHz	001
4.0 MHz - 8.0 MHz	010
8.0 MHz - 16 MHz	011*
CLK > 16 MHz	1xx
*denotes reset value	

Next, the gain of the ADC is selected. In single-ended mode, the maximum DC input voltage the ADC can accept is equal to VREF. If the internal voltage reference is used, this value is about 2.4 V. The maximum voltage that can be produced by the temp sensor is slightly more than 1 V. Therefore, we can safely set the ADC gain to '2' to increase the temperature resolution. The configuration bits that set the ADC gain are located in ADC0CF. Thus we have:

```

; set ADC clk = SYSCLK/8;
; set ADC gain = 2
mov ADC0CF, #61h

```

The remaining ADC configuration bits are located in ADC0CN, which is a bit addressable SFR. Any valid conversion start mechanism can be selected: Timer 2 or Timer 3 overflows, writing '1' to ADBUSY, or external CNVSTR. The software examples that follow



use Timer 3 overflows as the start of conversion source. Here, we use writing a '1' to ADBUSY.

We configure the ADC for low-power tracking mode, to use writing a '1' to ADBUSY as the start-of-conversion signal, and to output data in a left-justified format by writing the following:

```
; enable ADC;
; enable low-power tracking mode;
; clear pending conversion
; complete interrupts;
; select ADBUSY as start-of-
; conversion source;
; clear pending Window Compare
; interrupts;
; set output data format to
; left-justified.
mov ADC0CN, #0c1h
```

At this point, we can initiate a conversion by writing a '1' to ADBUSY:

```
setb ADBUSY ; start conversion
```

Now we wait for the conversion to complete:

```
; wait for conversion to complete
jnb ADCINT, $
```

Once the conversion is complete, the 16-bit value in the ADC output registers, ADC0H and ADC0L, contains a code which is proportional to the absolute temperature of the die in the device. The following section tells how to interpret the code to find the temperature in degrees Celsius.

## Interpreting the Results

The temp sensor produces a voltage output which is proportional to the absolute temperature of the die in the device. The relationship between this voltage and the temperature in degrees C is shown in Equation 1.

### Equation 1.

$$Vtemp = \left(2.86 \frac{mV}{C}\right) \times Temp + 776mV$$

Where:

*Vtemp* = the output voltage of the temp sensor in mV

*Temp* = the die temperature in degrees C

The transfer characteristic of the temp sensor is shown graphically in Figure 1.

The temp sensor voltage is not directly measurable outside the device. Instead, it is presented as one of the inputs of the ADC mux, allowing the ADC to measure the voltage and produce an output code which is proportional to it.

The code produced by the ADC in **left-justified** single-ended mode is proportional to the input voltage as follows:

### Equation 2.

$$CODE = Vin \times \frac{Gain}{VREF} \times 2^{16}$$

Where:

*CODE* = the left-justified ADC output code

*Gain* = the gain of the ADC's PGA

*VREF* = the value of the voltage reference, which is around 2.43 V if the internal VREF is used.

Substituting Equation 1 into Equation 2, assuming Gain=2 and VREF = 2.43V, solving for Temp and rearranging, we obtain an output



Temperature which in terms of CODE and a pair of constants:

### Equation 3.

$$Temp = \frac{(CODE - 41857)}{154}$$

Where:

Temp = the temperature in degrees C

CODE = the left-justified ADC output code.

## Implementation Considerations

### Self-Heating

The temp sensor measures the temperature of the die of the device, which is likely to be a few degrees warmer than the surrounding ambient temperature due to device power dissipation.

In order to find the ambient temperature, the temperature increase due to self-heating must be subtracted from the result. The value of this temperature increase can be calculated or measured.

There are many factors that contribute to the amount of device self-heating. Chief among

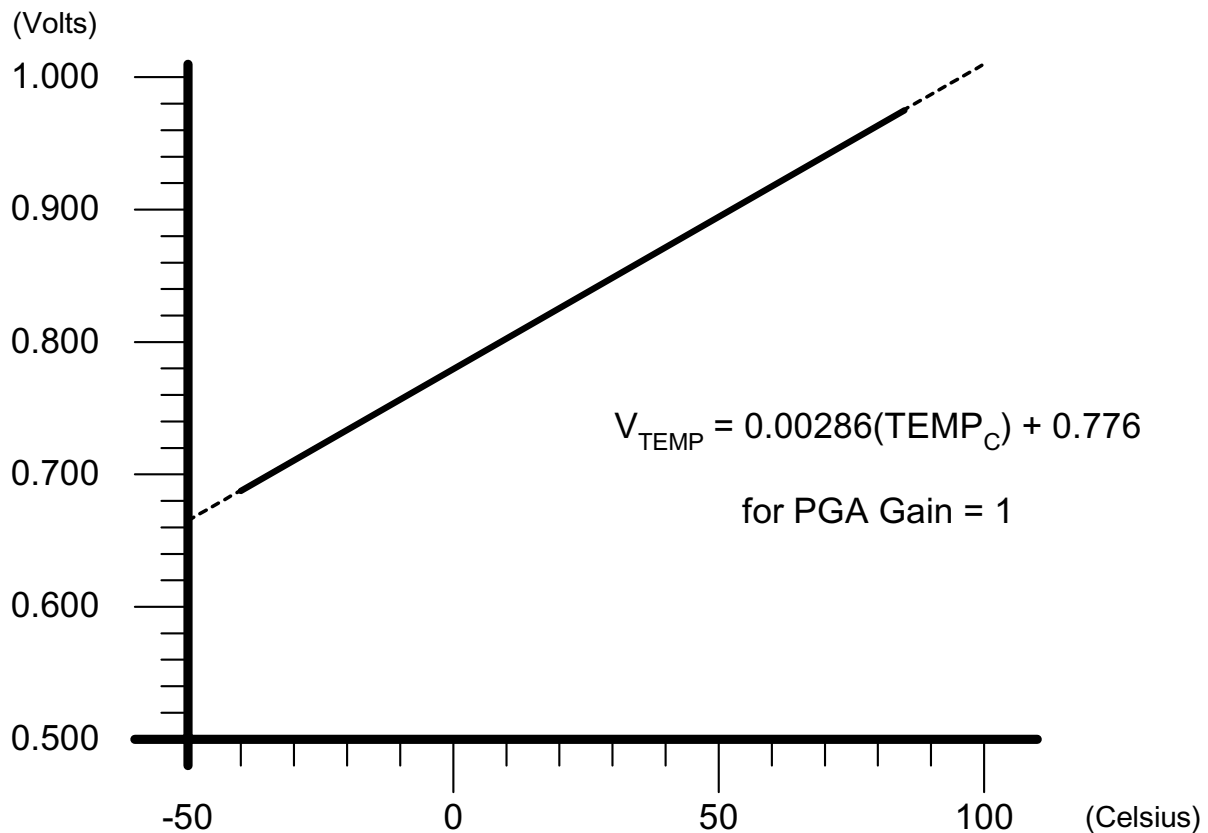


Figure 1. Temperature Sensor Transfer Characteristic



these are: power supply voltage, operating frequency, the thermal dissipation characteristics of the package, device mounting on the PCB, and airflow over the package. The temperature increase can be calculated to the first order by multiplying the device's power dissipation by the thermal dissipation constant of the package, usually called  $\theta_{JA}$ . The use of this constant assumes a standard PCB mounting, all pins soldered to traces, and no airflow over the package.

For a C8051F005 operating at 11.0592 MHz and a 3.3 V power supply, the power dissipation is approximately 35 mW. The  $\theta_{JA}$  value for the 64-pin TQFP package is 39.5 degrees C/W. This equates to a self-heating number of  $39.5 * 35e-3 \sim 1.4$  degrees C.

The temperature increase due to self-heating can be measured in a number of ways. One method is to initiate a conversion soon after applying power to the device to get a 'cold' temperature reading, then measure again after about a minute of operation, to get a 'hot' temperature reading. The difference between the two measurements is the contribution due to self-heating.

Another method is to operate the device from a low SYSCLK frequency, for example a 32 kHz watch crystal, and take a temperature measurement, then operate the device at a higher frequency, the 16 MHz internal oscillator for example, and take the difference. The amount of self-heating at the lower clock frequency is negligible because the power dissipation of the device at that frequency is low.

### **Averaging**

To minimize the effects of noise on the temperature conversion result, one technique is to 'oversample' the data and then average it. 'Oversampling' means that the sample rate of

the ADC is set higher than the required output word rate. As a rule-of-thumb, the output resolution increases by 1 bit for every power of 4 of oversampling.



## Example Code

### “Temp\_3.c”

```
//-----  
// Temp_3.c  
//-----  
// Copyright 2001 Cygnal Integrated Products, Inc.  
//  
// AUTH: BW  
// DATE: 4 SEP 01  
//  
// This program prints the C8051F0xx die temperature out the hardware  
// UART at 9600bps. Assumes an 18.432MHz crystal is attached between  
// XTAL1 and XTAL2.  
//  
// The ADC is configured to look at the on-chip temp sensor. The sampling  
// rate of the ADC is determined by the constant <SAMPLE_RATE>, which is given  
// in Hz.  
//  
// The ADC0 End of Conversion Interrupt Handler retrieves the sample  
// from the ADC and adds it to a running accumulator. Every <INT_DEC>  
// samples, the ADC updates and stores its result in the global variable  
// <temperature>, which holds the current temperature in hundredths of a  
// degree. The sampling technique of adding a set of values and  
// decimating them (posting results every (n)th sample) is called 'integrate  
// and dump.' It is easy to implement and requires very few resources.  
//  
// For each power of 4 of <INT_DEC>, you gain 1 bit of effective resolution.  
// For example, <INT_DEC> = 256 gain you 4 bits of resolution: 4^4 = 256.  
//  
// Also note that the ADC0 is configured for 'LEFT' justified mode. In this  
// mode, the MSB of the ADC word is located in the MSB position of the ADC0  
// high byte. Using the data in this way makes the magnitude of the resulting  
// code independent of the number of bits in the ADC (12- and 10-bits behave  
// the same).  
//  
// Target: C8051F00x or C8051F01x  
// Tool chain: KEIL C51 6.03 / KEIL EVAL C51  
//  
//-----  
// Includes  
//-----  
  
#include <c8051f000.h>           // SFR declarations  
#include <stdio.h>  
  
//-----  
// 16-bit SFR Definitions for 'F00x  
//-----  
  
sfr16 DP      = 0x82;           // data pointer  
sfr16 TMR3RL  = 0x92;           // Timer3 reload value  
sfr16 TMR3    = 0x94;           // Timer3 counter  
sfr16 ADC0    = 0xbe;           // ADC0 data
```



## AN003 - Using the On-Chip Temperature Sensor

```
sfr16 ADC0GT = 0xc4; // ADC0 greater than window
sfr16 ADC0LT = 0xc6; // ADC0 less than window
sfr16 RCAP2 = 0xca; // Timer2 capture/reload
sfr16 T2 = 0xcc; // Timer2
sfr16 DAC0 = 0xd2; // DAC0 data
sfr16 DAC1 = 0xd5; // DAC1 data

//-----
// Global CONSTANTS
//-----

#define BAUDRATE 9600 // Baud rate of UART in bps
#define SYSCLK 18432000 // SYSCLK frequency in Hz
#define SAMPLE_RATE 50000 // Sample frequency in Hz
#define INT_DEC 256 // integrate and decimate ratio

sbit LED = P1^6; // LED='1' means ON
sbit SW1 = P1^7; // SW1='1' means switch pressed

//-----
// Function PROTOTYPES
//-----

void SYSCLK_Init (void);
void PORT_Init (void);
void UART0_Init (void);
void ADC0_Init (void);
void Timer3_Init (int counts);
void ADC0_ISR (void);

//-----
// Global VARIABLES
//-----

long result; // ADC0 decimated value

//-----
// MAIN Routine
//-----

void main (void) {
    long temperature; // temperature in hundredths of a
                    // degree C
    int temp_int, temp_frac; // integer and fractional portions of
                          // temperature

    WDTCN = 0xde; // disable watchdog timer
    WDTCN = 0xad;

    SYSCLK_Init (); // initialize oscillator
    PORT_Init (); // initialize crossbar and GPIO
    UART0_Init (); // initialize UART0
    Timer3_Init (SYSCLK/SAMPLE_RATE); // initialize Timer3 to overflow at
    // sample rate

    ADC0_Init (); // init ADC
}
```

## AN003 - Using the On-Chip Temperature Sensor



```
ADCEN = 1;                // enable ADC

EA = 1;                   // Enable global interrupts

while (1) {
    EA = 0;                // disable interrupts
    temperature = result;
    EA = 1;                // re-enable interrupts

    // calculate temperature in hundredths of a degree
    temperature = temperature - 41857;
    temperature = (temperature * 100L) / 154;
    temp_int = temperature / 100;
    temp_frac = temperature - (temp_int * 100);
    printf ("Temperature is %+02d.%02d\n", temp_int, temp_frac);

    LED = SW1;             // LED reflects state of switch
}

//-----
// Initialization Subroutines
//-----

//-----
// SYSCLK_Init
//-----
//
// This routine initializes the system clock to use an 22.1184MHz crystal
// as its clock source.
//
void SYSCLK_Init (void)
{
    int i;                 // delay counter

    OSCXCN = 0x67;         // start external oscillator with
                          // 18.432MHz crystal

    for (i=0; i < 256; i++) ; // Wait for crystal osc. to start

    while (!(OSCXCN & 0x80)) ; // Wait for crystal osc. to settle

    OSCICN = 0x88;         // select external oscillator as SYSCLK
                          // source and enable missing clock
                          // detector
}

//-----
// PORT_Init
//-----
//
// Configure the Crossbar and GPIO ports
//
void PORT_Init (void)
{
    XBR0 = 0x04;           // Enable UART0
    XBR1 = 0x00;
}
```





## AN003 - Using the On-Chip Temperature Sensor

```
XBR2      = 0x40;           // Enable crossbar and weak pull-ups
PRT0CF |= 0x01;           // enable TX0 as a push-pull output
PRT1CF |= 0x40;           // enable P1.6 (LED) as push-pull output
}

//-----
// UART0_Init
//-----
//
// Configure the UART using Timer1, for <baudrate> and 8-N-1.
//
void UART0_Init (void)
{
    SCON      = 0x50;           // SCON: mode 1, 8-bit UART, enable RX
    TMOD      = 0x20;           // TMOD: timer 1, mode 2, 8-bit reload
    TH1       = -(SYSCLK/BAUDRATE/16); // set Timer1 reload value for baudrate
    TR1       = 1;             // start Timer1
    CKCON |= 0x10;           // Timer1 uses SYSCLK as time base
    PCON |= 0x80;           // SMOD = 1
    TI        = 1;           // Indicate TX ready
}

//-----
// ADC0_Init
//-----
//
// Configure ADC0 to use Timer3 overflows as conversion source, to
// generate an interrupt on conversion complete, and to use left-justified
// output mode. Enables ADC end of conversion interrupt. Leaves ADC disabled.
//
void ADC0_Init (void)
{
    ADC0CN = 0x05;           // ADC0 disabled; normal tracking
                                // mode; ADC0 conversions are initiated
                                // on overflow of Timer3; ADC0 data is
                                // left-justified
    REF0CN = 0x07;           // enable temp sensor, on-chip VREF,
                                // and VREF output buffer
    AMX0SL = 0x0f;           // Select TEMP sens as ADC mux output
    ADC0CF = 0x80;           // ADC conversion clock = SYSCLK/16
    ADC0CF |= 0x01;           // PGA gain = 2

    EIE2 |= 0x02;           // enable ADC interrupts
}

//-----
// Timer3_Init
//-----
//
// Configure Timer3 to auto-reload at interval specified by <counts> (no
// interrupt generated) using SYSCLK as its time base.
//
void Timer3_Init (int counts)
{
    TMR3CN = 0x02;           // Stop Timer3; Clear TF3;
                                // use SYSCLK as timebase
    TMR3RL = -counts;        // Init reload values
}
```

## AN003 - Using the On-Chip Temperature Sensor



```
TMR3      = 0xffff;           // set to reload immediately
EIE2     &= ~0x01;          // disable Timer3 interrupts
TMR3CN   |= 0x04;           // start Timer3
}

//-----
// Interrupt Service Routines
//-----

//-----
// ADC0_ISR
//-----
//
// ADC0 end-of-conversion ISR
// Here we take the ADC0 sample, add it to a running total <accumulator>, and
// decrement our local decimation counter <int_dec>. When <int_dec> reaches
// zero, we post the decimated result in the global variable <result>.
//
void ADC0_ISR (void) interrupt 15 using 1
{
    static unsigned int_dec=INT_DEC;    // integrate/decimate counter
                                        // we post a new result when
                                        // int_dec = 0
    static long accumulator=0L;        // here's where we integrate the
                                        // ADC samples

    ADCINT = 0;                        // clear ADC conversion complete
                                        // indicator

    accumulator += ADC0;                // read ADC value and add to running
                                        // total
    int_dec--;                           // update decimation counter

    if (int_dec == 0) {                 // if zero, then post result
        int_dec = INT_DEC;              // reset counter
        result = accumulator >> 8;
        accumulator = 0L;                // reset accumulator
    }
}
```



### "Temp\_2.asm"

```
-----  
; Temp_2.ASM  
-----  
; Copyright 2001, Cygnal Integrated Products, Inc.  
;  
; FILE:          Temp_2.ASM  
; DEVICE:        C8051F00x, C8051F01x  
; ASSEMBLER:     Keil A51  
; AUTH:          BW  
; DATE:          23 JUL 01  
;  
; This program provides an example of how to configure the on-chip temperature  
; sensor with the ADC.  The ADC is configured for left-justified mode, so this  
; code will work as-is on devices which have 10 or 12-bit ADCs.  
;  
; An external 18.432MHz crystal is used as the system clock source.  
;  
; The ADC is configured for left-justified mode, GAIN = 2, using Timer3 overflows  
; as the start-of-conversion source.  Timer3 is configured in auto-reload mode  
; to overflow every 10ms.  The ADC conversion complete interrupt handler  
; reads the ADC value and compares it with the expected value for room  
; temperature (about 25 degrees C), stored in ROOMCODE.  If the measured  
; temperature is below this value, the LED is turned off.  If the measured  
; value is above ROOMCODE, the LED is turned on.  
;  
; The LED switch point can be easily modified by changing the value of ROOMCODE.  
;  
-----  
;  
-----  
; EQUATES  
-----  
$INCLUDE (C8051F000.inc)  
  
LED          EQU          P1.6          ; LED on target board ('1' is LED ON)  
  
SYSCLK       EQU          18432         ; SYSCLK frequency in kHz  
  
TC_10ms     EQU          (SYSCLK / 12) * 10; number of timer counts in 10ms  
  
ROOMCODE    EQU          0xb3f0        ; left-justified ADC value for 25 degrees C.  
  
-----  
; VARIABLES  
-----  
  
MYDATA      SEGMENT DATA              ; declare DATA segment  
            RSEG MYDATA                ; select DATA segment  
  
; ADC data variables  
TEMPCODE:   DS          2              ; holding register for temp code (16-bit)  
; stored MSB-first (like in 'C' code)  
  
-----
```

## AN003 - Using the On-Chip Temperature Sensor



```
; STACK

STACK      SEGMENT IDATA      ; declare STACK segment
           RSEG  STACK
           DS    80h          ; reserve 128 bytes for stack

;-----
; MACRO DEFINITIONS
;-----

;-----
; RESET AND INTERRUPT VECTOR TABLE
;-----

CSEG AT 0
    ljmp  Main

    org  7bh
    ljmp  ADC0_ISR      ; ADC0 end of conversion interrupt

;-----
; MAIN PROGRAM CODE
;-----

Temp_2      SEGMENT  CODE      ; declare CODE segment
           RSEG  Temp_2      ; select CODE segment
           USING 0          ; using register bank 0

Main:
    mov  WDTCN, #0deh      ; disable watchdog timer
    mov  WDTCN, #0adh

    mov  SP, #STACK-1     ; init stack pointer

    mov  OSCXCN, #67h     ; enable external crystal oscillator
                       ; at 18.432MHz

    clr  A                ; wait at least 1ms
    djnz acc, $           ; wait 512us
    djnz acc, $           ; wait 512us

osc_wait:
    mov  a, OSCXCN
    jnb  acc.7, osc_wait  ; poll for XTLVLD-->1

    orl  OSCICN, #08h     ; select external oscillator as
                       ; system clock source
    orl  OSCXCN, #80h     ; enable missing clock detector

    mov  XBR2, #40h       ; Enable crossbar and weak pull-ups

    orl  PRT1CF, #40h     ; enable P1.6 (LED on target board) as
                       ; push-pull

    acall ADC0_Init      ; initialize ADC0 and temp sensor
    acall Timer3_Init    ; initialize Timer3
```



## AN003 - Using the On-Chip Temperature Sensor

```
    acall Timer3_Start          ; enable Timer3
    acall ADC0_Enable          ; enable ADC

    setb EA                    ; enable global interrupts

    sjmp $                      ; spin forever

;-----
; MAIN SUBROUTINES
;-----

;-----
; INTERRUPT VECTORS
;-----
;-----
; ADC0_ISR
;
; This ISR is activated on the completion of an ADC sample. When this event
; occurs, the ADC value is copied to the holding variable TEMPCODE, and is
; compared with the code for 25 degrees C. If the temperature is above
; 25 degrees C, the LED is turned on. If the temperature is below 25 degrees
; C, the LED is turned off. There is no correction here for self-heating.
;
ADC0_ISR:
    push PSW                    ; preserve registers
    push acc

    clr ADCINT                  ; clear ADC0 interrupt flag

    mov TEMPCODE, ADC0H         ; copy MSB of ADC0 result into
                                ; TEMPCODE
    mov TEMPCODE+1, ADC0L       ; copy LSB of ADC result into
                                ; TEMPCODE

    ; compare TEMPCODE with value expected for 25 degrees C

    ; if (TEMPCODE - ROOMDEG) < 0, then turn LED off, otherwise, turn it on.
    ; calculate TEMPCODE - ROOMREG and store in TEMPCODE (16-bit subtract)

    clr C
    mov a, TEMPCODE+1           ; subtract LSBs
    subb a, #LOW(ROOMCODE)
    mov TEMPCODE+1, a           ; store new LSB
    mov a, TEMPCODE             ; subtract MSBs (and carry)
    subb a, #HIGH(ROOMCODE)
    mov TEMPCODE, a            ; store new MSB

    setb LED                    ; turn LED on.
    jnc ADC0_ISR_END           ; exit if subtract result was positive,
    clr LED                     ; otherwise, turn LED off then exit

ADC0_ISR_END:
    pop acc
    pop PSW

    reti
```

## AN003 - Using the On-Chip Temperature Sensor



```
-----  
; SUBROUTINES  
-----  
;-----  
; Timer3_Init  
;-----  
; This routine initializes Timer3 in 16-bit auto-reload mode to overflow  
; at 100Hz using SYSCLK/12 as its time base. Exits with Timer3  
; stopped and Timer3 interrupts disabled.  
;  
Timer3_Init:  
    mov    TMR3CN, #00h           ; stop Timer3, clear TF3, use  
                                ; SYSCLK/12 as timebase  
    mov    TMR3RLH, #HIGH(-TC_10ms); init reload values  
    mov    TMR3RLL, #LOW(-TC_10ms)  
    mov    TMR3H, #0ffh          ; set to auto-reload immediately  
    mov    TMR3L, #0ffh  
    anl    EIE2, #NOT(01h)       ; disable Timer3 interrupts  
  
    ret  
  
;-----  
; TIMER3_Start  
;-----  
; This routine starts Timer3  
;  
Timer3_Start:  
    orl    TMR3CN, #04h          ; set TR3  
    ret  
  
;-----  
; ADC0_Init  
;-----  
; This routine initializes ADC0 for left-justified mode monitoring the  
; on-chip temp sensor at a gain of '2'. Leaves ADC in disabled state.  
;  
ADC0_Init:  
    clr    ADCEN                 ; disable ADC  
    mov    REF0CN, #07h          ; enable temp sensor, bias  
                                ; generator, and output  
                                ; buffer  
    mov    AMX0SL, #0fh          ; select TEMP sensor as ADC0  
                                ; input  
    mov    ADC0CF, #80h          ; set SAR clock to SYSCLK/16  
    orl    ADC0CF, #01h          ; PGA Gain = 2  
    mov    ADC0CN, #45h          ; ADC disabled, low power  
                                ; track mode, ADC0 conversions  
                                ; initiated by overflow on  
                                ; Timer3, left-justified data  
  
    ret  
  
;-----  
; ADC0_Enable  
;-----  
; This routine enables the ADC and ADC interrupts.  
;  
ADC0_Enable:
```



## AN003 - Using the On-Chip Temperature Sensor

---

```
setb  ADCEN                ; enable ADC
orl   EIE2, #02h          ; enable ADC EOC interrupt
ret
```

```
-----
; End of file.
```

END