

# CodeVisionAVR C Library Functions Reference

## CodeVisionAVR C 库函数介绍

译自 CodeVisionAVR C Compiler Help

目录:

1. Character Type Functions — 字符类型函数
2. Standard C Input/Output Functions — 标准输入输出函数
3. Standard Library Functions — 标准库和内存分配函数
4. Mathematical Functions — 数学函数
5. String Functions — 字符串函数
6. BCD Conversion Functions — BCD 转换函数
7. Memory Access Functions — 存储器访问函数
8. Delay Functions — 延时函数
9. LCD Functions — LCD 函数
10. LCD Functions for displays with 4x40 characters — 4×40 字符型 LCD 函数
11. LCD Functions for displays connected in 8 bit memory mapped mode — 以 8 位外部存储器模式接口的 LCD 显示函数
12. I2C Bus Functions — I2C 总线函数
13. National Semiconductor LM75 Temperature Sensor Functions — LM75 温度传感器函数
14. Dallas Semiconductor DS1621 Thermometer/Thermostat Functions — DS1621 温度计函数
15. Philips PCF8563 Real Time Clock Functions — PCF8563 实时时钟函数
16. Philips PCF8583 Real Time Clock Functions — PCF8583 实时时钟函数
17. Dallas Semiconductor DS1302 Real Time Clock Functions — DS1302 实时时钟函数
18. Dallas Semiconductor DS1307 Real Time Clock Functions — DS1307 实时时钟函数
19. 1 Wire Protocol Functions — 单线通讯协议函数
20. Dallas Semiconductor DS1820/DS1822 Temperature Sensors Functions — DS1820/1822 温度传感器函数
21. SPI Functions — SPI 函数
22. Power Management Functions — 电源管理函数
23. Gray Code Conversion Functions — 格雷码转换函数

前言:

如果你要使用库函数，就必须用 `#include` 包含相应的头文件。

例子:

```
/* 使用库函数前要先包含头文件 */
#include <math.h> // 有 abs 函数
#include <stdio.h> // 有 putsf 函数
void main(void) {
  int a,b;
  a=-99;
  /* 使用库函数 */
  b=abs(a);
  putsf("Hello world");
}
```

## 1. Character Type Functions — 字符类型函数

这些函数的原型放在“`..\INC`”目录的“`ctype.h`”头文件中。使用这些之前必须用“`#include`”包含头文件。

`unsigned char isalnum(char c)` — 如果 `c` 是数字或字母返回 1 。

`unsigned char isalpha(char c)` — 如果 `c` 是字母返回 1 。

`unsigned char isascii(char c)` — 如果 `c` 是 ASCII 码 (0...127) 返回 1 。

`unsigned char iscntrl(char c)` — 如果 `c` 是控制字符 (0..31 或 127) 返回 1 。

`unsigned char isdigit(char c)` — 如果 `c` 是数字返回 1 。

`unsigned char islower(char c)` — 如果 `c` 是小写字母返回 1 。

`unsigned char isprint(char c)` — 如果 `c` 是一个可打印字符 (32...127) 返回 1 。

`unsigned char ispunct(char c)` — 如果 `c` 是一个除空格、数字或字母的可打印字符返回 1 。

`unsigned char isspace(char c)` — 如果 `c` 是空格返回 1 。

`unsigned char isupper(char c)` — 如果 `c` 是大写字母返回 1 。

`unsigned char isxdigit(char c)` — 如果 `c` 是 16 进制数字返回 1 。

`char toascii(char c)` — 返回 `c` 对应的 ASCII 。

`unsigned char toint(char c)` — 把 `c` 当做 16 进制字符并返回对应的 10 进制数 (0...15)。

`char tolower(char c)` — 如果 `c` 是大写字母返回对应的小写字母。

`char toupper(char c)` — 如果 `c` 是小写字母返回对应的大写字母。

## 2. Standard C Input/Output Functions — 标准输入输出函数

这些函数的原型放在“`..\INC`”目录的“`stdio.h`”头文件中。使用这些之前必须用“`#include`”包含头文件。

`char getchar(void)` — 使用查询方式返回由 UART 接收的一个字符。

`void putchar(char c)` — 使用查询方式由 UART 发送一个字符 `c` 。

使用这些函数之前，你必须：

设置 UART 的波特率，设置接收允许，设置发送允许。

例子:

```
#include <90s8515.h>
#include <stdio.h>
/* 晶振频率 [Hz] */
#define xtal 4000000L
/* 波特率 */
#define baud 9600
void main(void) {
char k;
/* 设置波特率 */
UBRR=xtal/16/ baud-1;
/* 设置 UART 控制寄存器, RX & TX 允许, 不使用中断, 8 位数据模式 */
UCR=0x18;
while (1) {
/* 接收 */
k=getchar();
/* 发送 */
putchar(k);
};
}
```

你也可以使用 **Project|Configure|C Compiler** 菜单选项设置波特率。

如果你使用其它的输入输出外设, 你必须根据你的外设修改 `getchar` 和 `putchar` 函数。这些函数的源代码在 `stdio.h` 文件里。所有高级别的输入输出函数都使用 `getchar` 和 `putchar`。

`void puts(char *str)` — 使用 `putchar` 把 SRAM 中的以空字符结束的字符串输出, 并在后面加换行符。

`void putsf(char flash *str)` — 使用 `putchar` 把 FLASH 中的以空字符结束的字符串输出, 并在后面加换行符。

`void printf(char flash *fmtstr [ , arg1, arg2, ...])` — 使用 `putchar` 按格式说明符输出格式化文本 `fmtstr` 字符串。

格式化文本 `fmtstr` 字符串是常量, 必须放在 FLASH 中。

`printf` 执行的是标准 C 的一个子集。

下面是格式化说明符:

```
%c 输出一个 ASCII 字符
%d 输出有符号十进制整数
%i 输出有符号十进制整数
%u 输出无符号十进制整数
%x 输出小写字母的十六进制整数
%X 输出大写字母的十六进制整数
%s 输出 SRAM 中的以空字符结束的字符串
%% 输出 % 字符
```

所有输出的数都是右对齐的, 并在左侧加空格补齐。

如果在 % 和 `d`、`i`、`u`、`x` 或 `X` 之间加入一个字符 `0`, 那么输出的数的左侧加 `0` 补齐。

如果在 % 和 `d`、`i`、`u`、`x` 或 `X` 之间加入一个字符 `-`, 那么输出的数左对齐。

如果在 % 和 d、i、u、x 或 X 之间加入宽度限制符 (0..9)，可以指定输出的数的最小宽度。如果在宽度限制符前加入字符 -，输出的数左对齐。

```
void sprintf(char *str, char flash *fmtstr [, arg1, arg2, ...])
```

这个函数与 printf 类似，只是它的格式化字符放在以空字符结尾的字符串 str 中。

```
char *gets(char *str, unsigned char len) — 使用 getchar 接收以换行符结束的字符串 str。
```

换行符会被 0 替换。

字符串的最大长度是 len。如果已经收到了 len 个字符后还没有收到换行符，那么字符串就以 0 结束，函数停止执行并退出。

函数的返回值是指向 str 的指针。

```
signed char scanf(char flash *fmtstr [, arg1 address, arg2 address, ...]) — 使用 getchar 按格式说明符接收格式化文本 fmtstr 字符串。
```

格式化文本 fmtstr 字符串是常量，必须放在 FLASH 中。

scanf 执行的是标准 C 的一个子集。

下面是格式化说明符：

%c 接收一个 ASCII 字符

%d 接收有符号十进制整数

%i 接收有符号十进制整数

%u 接收无符号十进制整数

%x 接收无符号十六进制整数

%s 接收以空字符结束的字符串

函数返回成功接收的个数，如果返回-1 表示接收出错。

```
signed char sscanf(char *str, char flash *fmtstr [, arg1 address, arg2 address, ...])
```

这个函数与 scanf 类似，只是它的格式化字符放在 SRAM 中的以空字符结尾的字符串 str 中。

### 3. Standard Library Functions — 标准库和内存分配函数

这些函数的原型放在“..\INC”目录的“stdlib.h”头文件中。使用这些之前必须用“#include”包含头文件。

```
int atoi(char *str) — 转换字符串 str 为整型数并返回它的值，字符串 str 起始必须是十进制数字的字符，否则返回 0。当碰到字符串中第一个非十进制数字的字符时，转换结束。
```

```
long int atol(char *str) — 转换字符串 str 为长整型数并返回它的值，字符串 str 起始必须是长整型数形式字符，否则返回 0。
```

```
void itoa(int n, char *str) — 转换整型数 n 为字符串 str。
```

```
void ltoa(long int n, char *str) — 转换长整型数 n 为字符串 str。
```

```
void ftoa(float n, unsigned char decimals, char *str) — 转换浮点数 n 为字符串 str。
```

由 decimals 指定四舍五入保留小数位（最多五位）。

例子：

```
char *pi;
```

```
ftoa(3.1415926,3,pi);//pi[]="3.142"
```

```
void ftoe(float n, unsigned char decimals, char *str) — 转换浮点数 n 为字符串 str。
```

字符串表示为科学计数法形式，由 decimals 指定四舍五入保留小数位（最多五位）。

例子：

```
char *pi10;
```

```
ftoe(3.1415926*10,4,pi10);//pi10[]="3.1416e1"
```

`float atof(char *str)` — 转换字符串 `str` 为浮点数并返回它的值，字符串 `str` 起始必须是数字字符或小数点，否则返回 0。当碰到字符串中第一个十进制数字和小数点以外的字符时，转换结束。

`int rand(void)` — 产生一个 0 到 32767 之间的伪随机数。

`void srand(int seed)` — 设置伪随机数发生器的种子数。

#### 4. Mathematical Functions — 数学函数

这些函数的原型放在“..\INC”目录的“math.h”头文件中。使用这些之前必须用“#include”包含头文件。

`unsigned char cabs(signed char x)` — 返回 `x` 的绝对值。

`unsigned int abs(int x)` — 返回 `x` 的绝对值。

`unsigned long labs(long int x)` — 返回 `x` 的绝对值。

`float fabs(float x)` — 返回 `x` 的绝对值。

`signed char cmax(signed char a, signed char b)` — 返回 `a` 和 `b` 的最大值。

`int max(int a, int b)` — 返回 `a` 和 `b` 的最大值。

`long int lmax(long int a, long int b)` — 返回 `a` 和 `b` 的最大值。

`float fmax(float a, float b)` — 返回 `a` 和 `b` 的最大值。

`signed char cmin(signed char a, signed char b)` — 返回 `a` 和 `b` 的最小值。

`int min(int a, int b)` — 返回 `a` 和 `b` 的最小值。

`long int lmin(long int a, long int b)` — 返回 `a` 和 `b` 的最小值。

`float fmin(float a, float b)` — 返回 `a` 和 `b` 的最小值。

`signed char csign(signed char x)` — 当 `x` 分别为负数、0、正数时，返回 -1、0、1。

`signed char sign(int x)` — 当 `x` 分别为负数、0、正数时，返回 -1、0、1。

`signed char lsign(long int x)` — 当 `x` 分别为负数、0、正数时，返回 -1、0、1。

`signed char fsign(float x)` — 当 `x` 分别为负数、0、正数时，返回 -1、0、1。

`unsigned char isqrt(unsigned int x)` — 返回无符号整数 `x` 的平方根。

`unsigned int lsqrt(unsigned long x)` — 返回无符号长整数 `x` 的平方根。

`float sqrt(float x)` — 返回正浮点数 `x` 的平方根。

`float floor(float x)` — 返回不大于 `x` 的最大整数。

`float ceil(float x)` — 返回对应 `x` 的整数，小数部分四舍五入。

`float fmod(float x, float y)` — 返回 `x/y` 的余数。

`float modf(float x, float *ipart)` — 把浮点数 `x` 分解成整数部分和小数部分。整数部分存放在 `ipart` 指向的变量中，小数部分应大于或等于 0 而小于 1 并作为函数的返回值。

`float ldexp(float x, int expn)` — 返回  $x \times 2^{\text{expn}}$ 。

`float frexp(float x, int *expn)` — 把浮点数 `x` 分解成数字部分 `y`（尾数）和以 2 为底的指数 `n` 两个部分即  $x = y \times 2^n$ ，`y` 要大于等于 0.5 小于 1，`y` 值被函数返回而 `expn` 值存放在 `expn` 指向的变量中。

`float exp(float x)` — 返回  $e^x$  的值。

`float log(float x)` — 返回 `x` 的自然对数。

`float log10(float x)` — 返回以 10 为底的 `x` 的对数。

`float pow(float x, float y)` — 返回  $x^y$  的值。

`float sin(float x)` — 返回 `x` 的正弦函数值，`x` 为弧度。

`float cos(float x)` — 返回 `x` 的余弦函数值，`x` 为弧度。

`float tan(float x)` — 返回 `x` 的正切函数值，`x` 为弧度。

`float sinh(float x)` — 返回  $x$  的双曲正弦函数值,  $x$  为弧度。

`float cosh(float x)` — 返回  $x$  的双曲余弦函数值,  $x$  为弧度。

`float tanh(float x)` — 返回  $x$  的双曲正切函数值,  $x$  为弧度。

`float asin(float x)` — 返回  $x$  的反正弦函数值, 返回值为弧度, 范围在  $-\pi/2$  到  $\pi/2$  之间,  $x$  的值必须在  $-1$  到  $1$  之间。

`float acos(float x)` — 返回  $x$  的反余弦函数值, 返回值为弧度, 范围在  $0$  到  $\pi$  之间,  $x$  的值必须在  $-1$  到  $1$  之间。

`float atan(float x)` — 返回  $x$  的反正弦函数值, 返回值为弧度, 范围在  $-\pi/2$  到  $\pi/2$  之间。

`float atan2(float y, float x)` — 返回  $y/x$  的反正弦函数值, 返回值为弧度, 范围在  $-\pi$  到  $\pi$  之间。

## 5. String Functions — 字符串函数

这些函数的原型放在 “`..\INC`” 目录的 “`string.h`” 头文件中。使用这些之前必须用 “`#include`” 包含头文件。

字符串函数用于 SRAM 和 FLASH 中的字符串的操作。

`char *strcat(char *str1, char *str2)` — 拷贝 `str2` 到 `str1` 的结尾, 返回 `str1` 的指针。

`char *strcatf(char *str1, char flash *str2)` — 拷贝 FLASH 中的 `str2` 到 `str1` 的结尾, 返回 `str1` 的指针。

`char *strncat(char *str1, char *str2, unsigned char n)` — 拷贝 `str2` (不含结束符 `NULL`) 的  $n$  个字符到 `str1` 的结尾, 如果 `str2` 的长度比  $n$  小, 则只拷贝 `str2`, 返回 `str1` 的指针。

`char *strncatf(char *str1, char flash *str2, unsigned char n)` — 拷贝 FLASH 中的字符串 `str2` (不含结束符 `NULL`) 的  $n$  个字符到 `str1` 的结尾, 如果 `str2` 的长度比  $n$  小, 则只拷贝 `str2`, 返回 `str1` 的指针。

`char *strchr(char *str, char c)` — 在字符串 `str` 中搜索第一个出现的 `c`。如果成功, 返回匹配字符的指针; 如果没有搜索到匹配字符, 返回 `NULL`。

`char *strrchr(char *str, char c)` — 在字符串 `str` 中搜索最后一个出现的 `c`。如果成功, 返回匹配字符的指针; 如果没有搜索到匹配字符, 返回 `NULL`。

`signed char strpos(char *str, char c)` — 在字符串 `str` 中搜索第一个出现的 `c`。如果成功, 返回匹配字符在字符串中的位置; 如果没有搜索到匹配字符, 返回  $-1$ 。

`signed char strrpos(char *str, char c)` — 在字符串 `str` 中搜索最后一个出现的 `c`。如果成功, 返回匹配字符在字符串中的位置; 如果没有搜索到匹配字符, 返回  $-1$ 。

`signed char strcmp(char *str1, char *str2)` — 比较两个字符串。如果相同, 返回  $0$ ; 如果 `str1 > str2`, 返回值  $> 0$ ; 如果 `str1 < str2`, 返回值  $< 0$ 。

`signed char strcmpf(char *str1, char flash *str2)` — 比较 SRAM 中的字符串 `str1` 和 FLASH 中的字符串 `str2`。如果相同, 返回  $0$ ; 如果 `str1 > str2`, 返回值  $> 0$ ; 如果 `str1 < str2`, 返回值  $< 0$ 。

`signed char strncmp(char *str1, char *str2, unsigned char n)` — 比较两个字符串的前  $n$  的字符。如果相同, 返回  $0$ ; 如果 `str1 > str2`, 返回值  $> 0$ ; 如果 `str1 < str2`, 返回值  $< 0$ 。

`signed char strncmpf(char *str1, char flash *str2, unsigned char n)` — 比较 SRAM 中的字符串 `str1` 和 FLASH 中的字符串 `str2` 的前  $n$  个字符。如果相同, 返回  $0$ ; 如果 `str1 > str2`, 返回值  $> 0$ ; 如果 `str1 < str2`, 返回值  $< 0$ 。

`char *strcpy(char *dest, char *src)` — 拷贝字符串 `src` 到字符串 `dest`, 返回 `dest` 的指针。

`char *strcpyf(char *dest, char flash *src)` — 拷贝 FLASH 中的字符串 `src` 到 SRAM 中的字符串 `dest`, 返回 `dest` 的指针。

`char *strncpy(char *dest, char *src, unsigned char n)` — 拷贝字符串 `src` 的前  $n$  个字符到字

字符串 dest，返回 dest 的指针。

char \*strncpyf(char \*dest, char flash \*src, unsigned char n) — 拷贝 FLASH 中的字符串 src 的前 n 个字符到 SRAM 中的字符串 dest，返回 dest 的指针。

unsigned char strspn(char \*str, char \*set) — 在字符串 str 中搜索与字符串 set 不匹配的字符。如果搜索到不匹配，返回不匹配字符在 str 的位置；如果 set 的所有字符都匹配，返回字符串 str 的长度。

unsigned char strspnf(char \*str, char flash \*set) — 在 SRAM 中的字符串 str 中搜索与 FLASH 中的字符串 set 不匹配的字符。如果搜索到不匹配，返回不匹配字符在 str 的位置；如果 set 的所有字符都匹配，返回字符串 str 的长度。

unsigned char strcspn(char \*str, char \*set) — 在字符串 str 中搜索与字符串 set 匹配的字符。如果搜索到匹配，返回匹配字符在 str 的位置；如果没有匹配字符，返回字符串 str 的长度。

unsigned char strcspnf(char \*str, char flash \*set) — 在字符串 str 中搜索与 FLASH 中的字符串 set 匹配的字符。如果搜索到匹配，返回匹配字符在 str 的位置；如果没有匹配字符，返回字符串 str 的长度。

char \*strpbrk(char \*str, char \*set) — 在字符串 str 中搜索与字符串 set 匹配的字符。如果搜索到匹配，返回匹配字符的指针；如果没有匹配字符，返回 NULL。

char \*strpbrkf(char \*str, char flash \*set) — 在字符串 str 中搜索与 FLASH 中的字符串 set 匹配的字符。如果搜索到匹配，返回匹配字符的指针；如果没有匹配字符，返回 NULL。

char \*strrbrk(char \*str, char \*set) — 在字符串 str 中搜索与字符串 set 匹配的最后一个字符。如果搜索到匹配，返回匹配字符的指针；如果没有匹配字符，返回 NULL。

char \*strrbrkf(char \*str, char flash \*set) — 在 SRAM 中的字符串 str 中搜索与 FLASH 中的字符串 set 匹配的最后一个字符。如果搜索到匹配，返回匹配字符的指针；如果没有匹配字符，返回 NULL。

char \*strstr(char \*str1, char \*str2) — 在字符串 str1 中搜索与字符串 str2 匹配的子字符串。如果找到匹配的子字符串，返回 str1 中的子字符串的起始地址指针；否则返回 NULL。

char \*strstrf(char \*str1, char flash \*str2) — 在 SRAM 中的字符串 str1 中搜索与 FLASH 中的字符串 str2 匹配的子字符串。如果找到匹配的子字符串，返回 str1 中的子字符串的起始地址指针；否则返回 NULL。

unsigned char strlen(char \*str) — 返回字符串 str 的长度（范围 0...255）。

unsigned int \_strlen(char \*str) — 返回字符串 str 的长度（范围 0...65535）。这个函数只能用在 SMALL 模式下。

unsigned int strlenf(char flash \*str) — 返回 FLASH 中的字符串 str 的长度。

void \*memcpy(void \*dest, void \*src, unsigned char n) — TINY 模式

void \*memcpy(void \*dest, void \*src, unsigned int n) — SMALL 模式

拷贝 src 的 n 个字节到 dest。Dest 与 src 不能重叠。返回 dest 的指针。

void \*memcpyf(void \*dest, void flash \*src, unsigned char n) — TINY 模式

void \*memcpyf(void \*dest, void flash \*src, unsigned int n) — SMALL 模式

拷贝 FLASH 中的字符串 src 的 n 个字节到 dest。Dest 与 src 不能重叠。返回 dest 的指针。

void \*memccpy(void \*dest, void \*src, char c, unsigned char n) — TINY 模式

void \*memccpy(void \*dest, void \*src, char c, unsigned int n) — SMALL 模式

拷贝字符串 src 的 n 个字节到 dest，如果碰到字符 c 就停止。Dest 与 src 不能重叠。如

果最后一个拷贝的字符是 c 返回 NULL，否则返回指向 dest+n+1 的指针。

void \*memmove(void \*dest,void \*src, unsigned char n) — TINY 模式

void \*memmove(void \*dest,void \*src, unsigned int n) — SMALL 模式

拷贝 src 的 n 个字节到 dest。Dest 与 src 可以重叠。返回 dest 的指针。

void \*memchr(void \*buf, unsigned char c, unsigned char n) — TINY 模式

void \*memchr(void \*buf, unsigned char c, unsigned int n) — SMALL 模式

在 buf 的前 n 个字节中搜索字符 c。如果搜索到 c，就返回指向 c 的指针；否则返回 NULL。

signed char memcmp(void \*buf1,void \*buf2, unsigned char n) — TINY 模式

signed char memcmp(void \*buf1,void \*buf2, unsigned int n) — SMALL 模式

比较字符串 buf1 和 buf2 的前 n 个字节。当 buf1<buf2，buf1=buf2，buf1>buf2 时分别返回<0, 0, >0。

signed char memcompf(void \*buf1,void flash \*buf2, unsigned char n) — TINY 模式

signed char memcompf(void \*buf1,void flash \*buf2, unsigned int n) — SMALL 模式

比较 SRAM 中的字符串 buf1 和 FLASH 中的字符串 buf2，最多比较前 n 的字节。当 buf1<buf2，buf1=buf2，buf1>buf2 时分别返回<0, 0, >0。

void \*memset(void \*buf, unsigned char c, unsigned char n) — TINY 模式

void \*memset(void \*buf, unsigned char c, unsigned int n) — SMALL 模式

用字符 c 填充 buf 的前 n 个字节。返回指向 buf 的指针。

## 6. BCD Conversion Functions — BCD 转换函数

这些函数的原型放在“..\INC”目录的“bcd.h”头文件中。使用这些之前必须用“#include”包含头文件。

unsigned char bcd2bin(unsigned char n) — 把 BCD 码数 n 转换为二进制。

unsigned char bin2bcd(unsigned char n) — 把二进制数 n 转换为 BCD 码。n 必须为从 0 到 99。

## 7. Memory Access Functions — 存储器访问函数

这些函数的原型放在“..\INC”目录的“mem.h”头文件中。使用这些之前必须用“#include”包含头文件。

void pokeb(unsigned int addr, unsigned char data) — 把一个字节 data 写在 SRAM 中指定的 addr 地址。

void pokew(unsigned int addr, unsigned int data) — 把一个字 data 写在 SRAM 中指定的 addr 地址。低字节在 addr，高字节在 addr+1。

unsigned char peekb(unsigned int addr) — 在 SRAM 中指定的 addr 地址读一个字节。

unsigned int peekw (unsigned int addr) — 在 SRAM 中指定的 addr 地址读一个字节。低字节从 addr 读出，高字节从 addr+1 读出。

## 8. Delay Functions — 延时函数

这些函数的原型放在“..\INC”目录的“delay.h”头文件中。使用这些之前必须用“#include”包含头文件。

这些函数使用程序循环产生延时。

调用这些函数之前要关闭中断，否则会比预期的延时要长。

一定要在 [Project|Configure|C Compiler](#) 菜单中设定正确的时钟频率。

void delay\_us(unsigned int n) — n 个微秒的延时。n 必须是常数表达式。



`void delay_ms(unsigned int n)` — `n` 个毫秒的延时。`n` 必须是常数表达式。这个函数会每一个毫秒清一次看门狗。

例子:

```
void main(void) {
/* 关闭中断 */
#asm("cli")
/* 100ms 延时 */
delay_us(100);
/* ..... */
/* 10ms 延时 */
delay_ms(10);
/* 打开中断 */
#asm("sei")
/* ..... */
}
```

## 9. LCD Functions — LCD 函数

LCD 函数针对由日立 HD44780 或兼容芯片控制的字符型 LCD 模块，支持以下类型：  
1x8, 2x12, 3x12, 1x16, 2x16, 2x20, 4x20, 2x24, 2x40。

这些函数的原型放在“..\INC”目录的“lcd.h”头文件中。使用这些之前必须用“#include”包含头文件。

在包含头文件前你必须声明哪一个口与 LCD 模块通讯。

例子:

```
/* 使用 PORTC 连接 LCD 模块 */
#asm
.equ __lcd_port=0x15
#endasm
/* 包含头文件 */
#include <lcd.h>
/* 可以使用 lcd 函数 */
```

LCD 模块与单片机口线连接方式如下:

[LCD]	[AVR Port]
RS (pin4) -----	bit 0
RD (pin 5) -----	bit 1
EN (pin 6) -----	bit 2
DB4 (pin 11) ---	bit 4
DB5 (pin 12) ---	bit 5
DB6 (pin 13) ---	bit 6
DB7 (pin 14) ---	bit 7

你还需要连接 LCD 的电源和亮度控制电压。

低级的 LCD 函数:

`void _lcd_ready(void)` — 等待，直到 LCD 模块准备好接收数据。

在使用 `_lcd_write_data` 函数向 LCD 模块写数据前必须调用此函数。

`void _lcd_write_data(unsigned char data)` — 向 LCD 模块的命令寄存器写一个字节 `data`。

这个函数可以用来修改 LCD 的配置。

例子：

```
/* 显示光标 */
```

```
_lcd_ready();
```

```
_lcd_write_data(0xe);
```

`void lcd_write_byte(unsigned char addr, unsigned char data)` — 向 LCD 模块的字符发生器或显存写一个字节 `data`。

例子：

```
/* LCD 用户定义
```

```
  芯片类型： AT90S8515
```

```
  内存模式： SMALL
```

```
  数据堆栈： 128 字节
```

```
  2x16 字符 LCD
```

```
  连接 PORTC
```

```
[LCD]      [ PORTC]
```

```
1 GND —   GND
```

```
2 +5V —   VCC
```

```
3 VCC —   LCD HEADER Vo
```

```
4 RS —    PC0
```

```
5 RD —    PC1
```

```
6 EN —    PC2
```

```
11 D4 —   PC4
```

```
12 D5 —   PC5
```

```
13 D6 —   PC6
```

```
14 D7 —   PC7
```

```
*/
```

```
/* LCD 连接 PORTC */
```

```
#asm
```

```
.equ __lcd_port=0x15 ;PORTC
```

```
#endasm
```

```
/* 包含头文件 */
```

```
#include <lcd.h>
```

```
typedef unsigned char byte;
```

```
/* 自定义字符的点阵数据，一个指向右上角的箭头 */
```

```
flash byte char0[8]={
```

```
0b0000000,
```

```
0b0001111,
```

```
0b0000011,
```

```
0b0000101,
```

```
0b0001001,
```

```
0b0010000,
```

```
0b0100000,
```

```
0b1000000};
```

```
/* 定义自定义字符 */
```

```

void define_char(byte flash *pc,byte char_code)
{
byte i,a;
a=(char_code<<3) | 0x40;
for (i=0; i<8; i++) lcd_write_byte(a++,*pc++);
}
void main(void)
{
/* 初始化 2 行 16 列 LCD */
lcd_init(16);
/* 定义自定义字符 0 */
define_char(char0,0);
/* 写显存 */
lcd_gotoxy(0,0);
lcd_putsf("User char 0:");
/* 显示自定义字符 0 */
lcd_putchar(0);
while (1); /* 死循环 */
}

```

unsigned char lcd\_read\_byte(unsigned char addr) — 从 LCD 模块的字符发生器或显存读出一个字节。

高级 LCD 函数：

void lcd\_init(unsigned char lcd\_columns) — 初始化 LCD 模块，清屏并把显示坐标设定在 0 列 0 行。LCD 模块的列必须指定（例如：16）。这时不显示光标。在使用其它高级 LCD 函数前，必须先调用此函数。

void lcd\_clear(void) — 清屏并把显示坐标设定在 0 列 0 行。

void lcd\_gotoxy(unsigned char x, unsigned char y) — 设定显示坐标在 x 列 y 行。列、行由 0 开始。

void lcd\_putchar(char c) — 在当前坐标显示字符 c 。

void lcd\_puts(char \*str) — 在当前坐标显示 SRAM 中的字符串 str 。

void lcd\_putsf(char flash \*str) — 在当前坐标显示 FLASH 中的字符串 str 。

## 10. LCD Functions for displays with 4x40 characters — 4×40 字符型 LCD 函数

只有商业版的 CodeVisionAVR C Compiler 才有这部分功能。

LCD 函数针对由日立 HD44780 或兼容芯片控制的字符型 4x40 LCD 模块。

这些函数的原型放在“..\INC”目录的“lcd4x40.h”头文件中。使用这些之前必须用“#include”包含头文件。

在包含头文件前你必须声明哪一个口与 LCD 模块通讯。

例子：

```

/*使用 PORTC 连接 LCD 模块*/
#asm
    .equ __lcd_port=0x15
#endasm
/* 包含头文件 */

```

```
#include <lcd4x40.h>
```

LCD 模块与单片机口线连接方式如下：

[LCD]	[AVR Port]
RS (pin 11) ---	bit 0
RD (pin 10) ---	bit 1
EN1 (pin 9) ----	bit 2
EN2 (pin 15) --	bit 3
DB4 (pin 4) ----	bit 4
DB5 (pin 3) ----	bit 5
DB6 (pin 2) ----	bit 6
DB7 (pin 1) ----	bit 7

你还需要连接 LCD 的电源和亮度控制电压。

低级的 LCD 函数：

`void _lcd_ready(void)` — 等待，直到 LCD 模块准备好接收数据。

在使用 `_lcd_write_data` 函数向 LCD 模块写数据前必须调用此函数。

`void _lcd_write_data(unsigned char data)` — 向 LCD 模块的命令寄存器写一个字节 `data`。  
这个函数可以用来修改 LCD 的配置。

在调用 `_lcd_ready` 和 `_lcd_write_data` 函数之前，全局变量 `_en1_msk` 必须设为 `LCD_EN1` (`LCD_EN2`)，以选择使用上半部（下半部）的 LCD 控制器。

例子：

```
/* 使用上半部 LCD */
_en1_msk=LCD_EN1;
_lcd_ready();
_lcd_write_data(0xe);
```

`void lcd_write_byte(unsigned char addr, unsigned char data)` — 向 LCD 模块的字符发生器或显存写一个字节 `data`。

`unsigned char lcd_read_byte(unsigned char addr)` — 从 LCD 模块的字符发生器或显存读出一个字节。

高级 LCD 函数：

`void lcd_init(void)` — 初始化 LCD 模块，清屏并把显示坐标设定在 0 列 0 行。LCD 模块的列必须指定（例如：16）。这时不显示光标。在使用其它高级 LCD 函数前，必须先调用此函数。

`void lcd_clear(void)` — 清屏并把显示坐标设定在 0 列 0 行。

`void lcd_gotoxy(unsigned char x, unsigned char y)` — 设定显示坐标在 x 列 y 行。列、行由 0 开始。

`void lcd_putchar(char c)` — 在当前坐标显示字符 `c`。

`void lcd_puts(char *str)` — 在当前坐标显示 SRAM 中的字符串 `str`。

`void lcd_putsf(char flash *str)` — 在当前坐标显示 FLASH 中的字符串 `str`。

## 11. LCD Functions for displays connected in 8 bit memory mapped mode — 以 8 位外部存储器模式接口的 LCD 显示函数

LCD 函数针对由日立 HD44780 或兼容芯片控制的字符型 LCD 模块。

LCD 作为一个 8 位的外设接在 AVR 的外部数据地址总线上。

这种接口方式可以用在 STK200 和 STK300 开发板上。LCD 的连接方式参考开发板的说明。

这些函数只能用于能外扩存储器的 AVR 芯片：AT90S4414，AT90S8515，ATmega603，ATmega103 和 ATmega161。

这些函数的原型放在“..\INC”目录的“lcdstk.h”头文件中。使用这些之前必须用“#include”包含头文件。

在 lcdstk.h 中支持以下类型的字符行 LCD 模块：1x8，2x12，3x12，1x16，2x16，2x20，4x20，2x24，2x40。

void \_lcd\_ready(void) — 等待，直到 LCD 模块准备好接收数据。

在使用宏\_LCD\_RS0 和\_LCD\_RS1\_向 LCD 模块写数据前必须调用此函数。

例子：

```
/* 允许显示光标 */
_lcd_ready();
_LCD_RS0=0xe;
```

宏\_LCD\_RS0（或\_LCD\_RS1）用来设 RS=0（或 RS=1）并访问 LCD 指令寄存器。

void lcd\_write\_byte(unsigned char addr, unsigned char data) — 向 LCD 的字符发生器或显存写一个字节。

例子：

```
/* LCD 用户定义
   芯片类型： AT90S8515
   内存模式： SMALL
   数据堆栈： 128 字节
   2x16 字符 LCD
   LCD 接在 STK200 的 LCD 接口 */
/* 包含头文件 */
#include <lcdstk.h>
typedef unsigned char byte;
/* 自定义字符的点阵数据，一个指向右上角的箭头 */
flash byte char0[8]={
0b0000000,
0b0001111,
0b0000011,
0b0000101,
0b0001001,
0b0010000,
0b0100000,
0b1000000};
/* 定义自定义字符 */
void define_char(byte flash *pc,byte char_code)
{
byte i,a;
a=(char_code<<3) | 0x40;
for (i=0; i<8; i++) lcd_write_byte(a++,*pc++);
```

```

}
void main(void)
{
/* 初始化 2 行 16 列 LCD */
lcd_init(16);
/* 定义自定义字符 0 */
define_char(char0,0);
/* 写显存 */
lcd_gotoxy(0,0);
lcd_putsf("User char 0:");
/* 显示自定义字符 0 */
lcd_putchar(0);
while (1); /* 死循环 */
}

```

`unsigned char lcd_read_byte(unsigned char addr)` — 从 LCD 模块的字符发生器或显存读出一个字节。

高级 LCD 函数:

`void lcd_init(unsigned char lcd_columns)` — 初始化 LCD 模块，清屏并把显示坐标设定在 0 列 0 行。LCD 模块的列必须指定（例如：16）。这时不显示光标。在使用其它高级 LCD 函数前，必须先调用此函数。

`void lcd_clear(void)` — 清屏并把显示坐标设定在 0 列 0 行。

`void lcd_gotoxy(unsigned char x, unsigned char y)` — 设定显示坐标在 x 列 y 行。列、行由 0 开始。

`void lcd_putchar(char c)` — 在当前坐标显示字符 c 。

`void lcd_puts(char *str)` — 在当前坐标显示 SRAM 中的字符串 str 。

`void lcd_putsf(char flash *str)` — 在当前坐标显示 FLASH 中的字符串 str 。

## 12. I2C Bus Functions — I2C 总线函数

这些函数的原型放在“..\INC”目录的“i2c.h”头文件中。使用这些之前必须用“#include”包含头文件。

利用这些函数可以把单片机作为主机或从机。

包含头文件之前，你必须先声明那些口线用于 I2C 总线。

例子:

```

/* 使用 PORTB 作 I2C 总线 */
/* SDA 为 PB3 */
/* SCL 为 PB4 */
#asm
    .equ __i2c_port=0x18
    .equ __sda_bit=3
    .equ __scl_bit=4
#endasm
/* 包含头文件*/
#include <i2c.h>

```

void i2c\_init(void) — 初始化 I2C 总线。调用其它 I2C 函数之前必须先调用此函数。

unsigned char i2c\_start(void) — 发送 START 信号。如果总线空闲，返回 1；如果总线忙，返回 0。

void i2c\_stop(void) — 发送 STOP 信号。

unsigned char i2c\_read(unsigned char ack) — 读一个字节。

The ack parameter specifies if an acknowledgement is to be issued after the byte was read.

Set ack to 0 for no acknowledgement or 1 for acknowledgement.

unsigned char i2c\_write(unsigned char data) — 写一个字节。如果从机有应答，返回 1；如果从机不应答，返回 0。

一个访问 Atmel 24C02—256 字节 EEPROM 的例子：

```
/* 使用 PORTB 作 I2C 总线 */
/* SDA 为 PB3 */
/* SCL 为 PB4 */
#asm
    .equ __i2c_port=0x18
    .equ __sda_bit=3
    .equ __scl_bit=4
#endasm
/* 包含头文件 */
#include <i2c.h>
#include <delay.h>
#define EEPROM_BUS_ADDRESS 0xa0
/* 从 EEPROM 读一个字节 */
unsigned char eeprom_read(unsigned char address) {
    unsigned char data;
    i2c_start();
    i2c_write(EEPROM_BUS_ADDRESS);
    i2c_write(address);
    i2c_start();
    i2c_write(EEPROM_BUS_ADDRESS | 1);
    data=i2c_read(0);
    i2c_stop();
    return data;
}
/* 向 EEPROM 写一个字节 */
void eeprom_write(unsigned char address, unsigned char data) {
    i2c_start();
    i2c_write(EEPROM_BUS_ADDRESS);
    i2c_write(address);
    i2c_write(data);
    i2c_stop();
    /* 10 延时等待写操作完成 */
    delay_ms(10);
}
```

```

void main(void) {
unsigned char i;
/* 初始化 I2C 总线 */
i2c_init();
/* 在地址 AAh 写入 55h */
eeprom_write(0xaa,0x55);
/* 从地址 AAh 读一个字节 */
i=eeprom_read(0xaa);
while (1); /* 死循环 */
}

```

### 13. National Semiconductor LM75 Temperature Sensor Functions — LM75 温度传感器函数

这些函数的原型放在“..\INC”目录的“lm75.h”头文件中。使用这些之前必须用“#include”包含头文件。

I2C 总线函数原型在 lm75.h 中自动包含。

包含头文件之前，你必须先声明那些口线通过 I2C 总线与 LM75 通讯。

例子：

```

/* 使用 PORTB 作 I2C 总线 */
/* SDA 为 PB3 */
/* SCL 为 PB4 */
#asm
    .equ __i2c_port=0x18
    .equ __sda_bit=3
    .equ __scl_bit=4
#endasm
/* 包含头文件 */
#include <lm75.h>

```

void lm75\_init(unsigned char chip,signed char thyst,signed char tos, unsigned char pol) — 初始化 LM75。

调用这个函数之前，必须调用函数 i2c\_init 初始化 I2C 总线。

调用其它的 LM75 函数之前，必须先调用此函数。

如果有多个 LM75 接在同一个 I2C 总线上，则每个 LM75 都要作初始化。

最多 8 只 LM75 可以接在同一个 I2C 总线上，地址为 0 到 7。

LM75 被设置为比较模式，就象一个恒温器。

O.S.输出脚在温度高于 tos 时激活，低于 thyst 时恢复高阻。

thyst 和 tos 都是摄氏度。

pol 设置 LM75 的 O.S. 输出脚在激活时的极性。如果 pol 是 0，输出激活时为低；如果 pol 是 1，输出激活时为高。

int lm75\_temperature\_10(unsigned char chip) — 读地址为 chip 的 LM75 的温度。温度是摄氏度，其值为返回值除以 10。

下面是如何显示地址分别为 0 和 1 的两个 LM75 的温度的例子：

```

/* 使用 PORTB 作 I2C 总线 */
/* SDA 为 PB3 */
/* SCL 为 PB4 */

```



```
#asm
    .equ __i2c_port=0x18
    .equ __sda_bit=3
    .equ __scl_bit=4
#endasm
/* 包含 LM75 头文件 */
#include <lm75.h>
/* LCD 模块在 PORTC */
#asm
    .equ __lcd_port=0x15
#endasm
/* 包含 LCD 头文件 */
#include <lcd.h>
/* 包含 sprintf 的函数原型*/
#include <stdio.h>
/*包含 abs 的函数原型*/
#include <math.h>
char display_buffer[33];
void main(void) {
    int t0,t1;
    /* 初始化 LCD, 2 行 16 列 */
    lcd_init(16);
    /* 初始化 I2C 总线 */
    i2c_init();
    /* 初始化地址 0 的 LM75 */
    /* thyst=20 度 tos=25 度 */
    lm75_init(0,20,25,0);
    /* 初始化地址 1 的 LM75 */
    /* thyst=30 度 tos=35 度 */
    lm75_init(1,30,35,0);
    /* 循环显示温度 */
    while (1)
    {
        /* 读地址 0 的温度 */
        t0=lm75_temperature_10(0);
        /* 读地址 1 的温度 */
        t1=lm75_temperature_10(1);
        /* 准备要显示的温度在 display_buffer */
        sprintf(display_buffer,"t0=%-i.%-u%cC\nt1=%-i.%-u%cC",
            t0/10,abs(t0%10),0xdf,t1/10,abs(t1%10),0xdf);
        /* 显示温度 */
        lcd_clear();
        lcd_puts(display_buffer);
    };
}
```

```
}

```

#### 14. Dallas Semiconductor DS1621 Thermometer/Thermostat Functions — DS1621 温度计函数

这些函数的原型放在“.\INC”目录的“ds1621.h”头文件中。使用这些之前必须用“#include”包含头文件。

I2C 总线函数原型在 ds1621.h 中自动包含。

包含头文件之前，你必须先声明那些口线通过 I2C 总线与 ds1621 通讯。

例子：

```
/* 使用 PORTB 作 I2C 总线 */
/* SDA 为 PB3 */
/* SCL 为 PB4 */
#asm
    .equ __i2c_port=0x18
    .equ __sda_bit=3
    .equ __scl_bit=4
#endasm
/* 包含 DS1621 头文件 */
#include <ds1621.h>

```

```
void ds1621_init(unsigned char chip,signed char tlow,signed char thigh, unsigned char pol)

```

— 初始化 DS1621。

调用这个函数之前，必须调用函数 i2c\_init 初始化 I2C 总线。

调用其它的 DS1621 函数之前，必须先调用此函数。

如果有多个 DS1621 接在同一个 I2C 总线上，则每个 DS1621 都要作初始化。

最多 8 只 DS1621 可以接在同一个 I2C 总线上，地址为 0 到 7。

除了测量温度 DS1621 函数也可以象恒温器一样工作。

Tout 输出脚在温度高于 thigh 时激活，低于 tlow 时恢复高阻。

tlow 和 thigh 都是摄氏度。

pol 设置 DS1621 的 Tout 输出脚在激活时的极性。如果 pol 是 0，输出激活时为低；如果 pol 是 1，输出激活时为高。

```
unsigned char ds1621_get_status(unsigned char chip) — 读出对应地址 chip 的 DS1621 的
配置/状态字节，其内容参看 DS1621 的数据手册。
```

```
void ds1621_set_status(unsigned char chip, unsigned char data) — 设置对应地址 chip 的
DS1621 的配置/状态字节，其内容参看 DS1621 的数据手册。
```

```
void ds1621_start(unsigned char chip) — 使地址为 chip 的 ds1621 从省电模式中退出，
开始温度测量和比较。
```

```
void ds1621_stop(unsigned char chip) — 使地址为 chip 的 DS1621 停止开始温度测量和，
并进入省电模式。
```

```
int ds1621_temperature_10(unsigned char chip) — 读地址为 chip 的 DS1621 的温度。温
度是摄氏度，其值为返回值除以 10。
```

下面是如何显示地址分别为 0 和 1 的两个 DS1621 的温度的例子：

```
/* 使用 PORTB 作 I2C 总线 */
/* SDA 为 PB3 */

```

```
/* SCL 为 PB4 */
#asm
    .equ __i2c_port=0x18
    .equ __sda_bit=3
    .equ __scl_bit=4
#endasm
/* 包含 DS1621 头文件 */
#include <ds1621.h>
/* LCD 模块在 PORTC */
#asm
    .equ __lcd_port=0x15
#endasm
/* 包含 LCD 头文件 */
#include <lcd.h>
/* 包含 sprintf 的函数原型*/
#include <stdio.h>
/*包含 abs 的函数原型*/
#include <math.h>
char display_buffer[33];
void main(void) {
    int t0,t1;
    /* 初始化 LCD, 2 行 16 列 */
    lcd_init(16);
    /* 初始化 I2C 总线 */
    i2c_init();
    /* 初始化地址 0 的 DS1621 */
    /* tlow=20 度 thigh=25 度 */
    ds1621_init(0,20,25,0);
    /* 初始化地址 1 的 DS1621 */
    /* tlow=30 度 thigh=35 度 */
    ds1621_init(1,30,35,0);
    /* 循环显示温度 */
    while (1)
    {
        /* 读地址 0 的温度 */
        t0=ds1621_temperature_10(0);
        /* 读地址 1 的温度 */
        t1=ds1621_temperature_10(1);
        /* 准备要显示的温度在 display_buffer */
        sprintf(display_buffer,"t0=%-i.%-u%cC\n\t1=%-i.%-u%cC",
            t0/10,abs(t0%10),0xdf,t1/10,abs(t1%10),0xdf);
        /* 显示温度 */
        lcd_clear();
        lcd_puts(display_buffer);
    }
}
```

```
};
}
```

## 15. Philips PCF8563 Real Time Clock Functions — PCF8563 实时时钟函数

只有商业版的 CodeVisionAVR C Compiler 才有这部分功能。

这些函数的原型放在 “..\INC” 目录的 “pcf8563.h” 头文件中。使用这些之前必须用 “#include” 包含头文件。

I2C 总线函数原型在 pcf8563.h 中自动包含。

包含头文件之前，你必须先声明那些口线通过 I2C 总线与 PCF8563 通讯。

例子：

```
/* 使用 PORTB 作 I2C 总线 */
/* SDA 为 PB3 */
/* SCL 为 PB4 */
#asm
    .equ __i2c_port=0x18
    .equ __sda_bit=3
    .equ __scl_bit=4
#endasm
/* 包含 PCF8563 头文件 */
#include <pcf8563.h>
```

void rtc\_init(unsigned char ctrl2, unsigned char clkout, unsigned char timer\_ctrl) — 初始化 PCF8563。

调用这个函数之前，必须调用函数 i2c\_init 初始化 I2C 总线。

调用其它的 PCF8563 函数之前，必须先调用此函数。

I2C 总线只能接一个 PCF8583。

参数 ctrl2 设定 PCF8563 的 Control/Status 2（控制/状态 2）寄存器的初值。

pcf8563.h 头文件定义了以下一些宏，以便设置 ctrl2 参数。

RTC\_TIE\_ON 置 Control/Status 2 寄存器的 TIE 位为 1

RTC\_AIE\_ON 置 Control/Status 2 寄存器的 AIE 位为 1

RTC\_TP\_ON 置 Control/Status 2 寄存器的 TI/TP 位为 1

这些宏可以用 | 操作符连在一起使用以同时设置多个位为 1。

参数 clkout 设定 PCF8563 CLKOUT Frequency（输出频率）寄存器的初值。

pcf8563.h 头文件定义了以下一些宏，以便设置 clkout 参数：

RTC\_CLKOUT\_OFF 关闭 PCF8563 的脉冲输出

RTC\_CLKOUT\_1 1Hz 脉冲输出

RTC\_CLKOUT\_32 32Hz 脉冲输出

RTC\_CLKOUT\_1024 1024Hz 脉冲输出

RTC\_CLKOUT\_32768 32768Hz 脉冲输出

参数 timer\_ctrl 设定了 PCF8563 的 Timer Control（定时器控制）寄存器的初值

pcf8563.h 头文件定义了一些宏方便设置 the timer\_ctrl 参数：

RTC\_TIMER\_OFF 关闭 PCF8563 倒数计时器

RTC\_TIMER\_CLK\_1\_60 设置 PCF8563 倒数计时器的时钟频率为 1/60Hz

RTC\_TIMER\_CLK\_1 设置 PCF8563 倒数计时器的时钟频率为 1Hz

RTC\_TIMER\_CLK\_64        设置 PCF8563 倒计数定时器的时钟频率为 64Hz

RTC\_TIMER\_CLK\_4096      设置 PCF8563 倒计数定时器的时钟频率为 4096Hz.

unsigned char rtc\_read(unsigned char address) — 从 PCF8563 的 address 地址的寄存器读出一个字节。

void rtc\_write(unsigned char address, unsigned char data) — 写一个字节到 PCF8563 的 address 地址的寄存器。

unsigned char rtc\_get\_time(unsigned char \*hour, unsigned char \*min, unsigned char \*sec) — 返回 RTC（实时时钟）的时间。

指针\*hour、\*min 和\*sec 必须指向接收小时、分钟和秒的变量。

如果函数返回 1，说明读的时间正确。如果函数返回 0，说明供电电压太低，时间不正确。

例子：

```
#asm
    .equ __i2c_port=0x18
    .equ __sda_bit=3
    .equ __scl_bit=4
#endasm
#include <pcf8563.h>
void main(void) {
    unsigned char ok,h,m,s;
    /* 初始化 I2C 总线 */
    i2c_init();
    /* 初始化 RTC，定时器中断允许，闹铃中断允许，
       CLKOUT 频率=1Hz，定时器时钟频率=1Hz */
    rtc_init(RTC_TIE_ON | RTC_AIE_ON,RTC_CLKOUT_1,RTC_TIMER_CLK_1);
    /* 从 RTC 读时间 */
    ok=rtc_get_time(&h,&m,&s);
    /* ..... */
}
```

void rtc\_set\_time(unsigned char hour, unsigned char min, unsigned char sec) — 设置 RTC 的时间。参数 hour, min 和 sec 对应时、分、秒。

void rtc\_get\_date(unsigned char \*date, unsigned char \*month, unsigned \*year) — 读取 RTC 的日历。指针\*date, \*month 和\*year 指向接收日、月、年的变量。

void rtc\_set\_date(unsigned char date, unsigned char month, unsigned year) — 设置 RTC 的日历。

void rtc\_alarm\_off(void) — 关闭 RTC 的闹铃功能。

void rtc\_alarm\_on(void) — 打开 RTC 的闹铃功能。

void rtc\_get\_alarm(unsigned char \*date, unsigned char \*hour, unsigned char \*min) — 读取 RTC 的闹铃的日期和时间。指针\*date, \*hour 和\*min 指向接收日期、小时、分钟的变量。

void rtc\_set\_alarm(unsigned char date, unsigned char hour, unsigned char min) — 设置 RTC 的闹铃的日期和时间。参数 date, hour 和 min 对应日期、小时、分钟。如果 date 是 0，这个参数将被忽略。调用这个函数后，闹铃被关闭。要调用 rtc\_alarm\_on 函数打开闹铃功能。

void rtc\_set\_timer(unsigned char val) — 设置 PCF8563 定时器的值。

## 16. Philips PCF8583 Real Time Clock Functions — PCF8583 实时时钟函数

只有商业版的 CodeVisionAVR C Compiler 才有这部分功能。

这些函数的原型放在 “..\INC” 目录的 “pcf8583.h” 头文件中。使用这些之前必须用 “#include” 包含头文件。

I2C 总线函数原型在 pcf8583.h 中自动包含。

包含头文件之前，你必须先声明那些口线通过 I2C 总线与 PCF8583 通讯。

例子：

```
/* 使用 PORTB 作 I2C 总线 */
/* SDA 为 PB3 */
/* SCL 为 PB4 */
#asm
    .equ __i2c_port=0x18
    .equ __sda_bit=3
    .equ __scl_bit=4
#endasm
/* 包含 PCF8583 头文件 */
#include <pcf8583.h>
```

void rtc\_init(unsigned char chip, unsigned char dated\_alarm) — 初始化 PCF8583。

调用这个函数之前，必须调用函数 i2c\_init 初始化 I2C 总线。

调用其它的 PCF8583 函数之前，必须先调用此函数。

如果 I2C 总线上接有多个 PCF8583，你必须通过参数 chip 对每一个 PCF8583 都初始化。

I2C 总线上最多可以接 2 个 PCF8583，它们的地址是 0 或 1。

参数 dated\_alarm 设定 RTC 闹铃是日期和时间同时起作用 (dated\_alarm=1)，还是只有时间起作用 (dated\_alarm=0)。

调用这个函数后 RTC 闹铃是关闭的。

unsigned char rtc\_read(unsigned char chip, unsigned char address) — 读 PCF8583 的 SRAM 中的一个字节。

void rtc\_write(unsigned char chip, unsigned char address, unsigned char data) — 向 PCF8583 的 SRAM 中写一个字节。

在向 SRAM 中写数据时，地址 10h 和 11h 存放的是年份的值。

unsigned char rtc\_get\_status(unsigned char chip) — 返回 PCF8583 的控制/状态寄存器的值。

调用这个函数时，全局变量 \_\_rtc\_status 和 \_\_rtc\_alarm 自动更新。变量 \_\_rtc\_status 存放着控制/状态寄存器的值。

变量 \_\_rtc\_alarm 如果为 1 表示有闹铃。

void rtc\_get\_time(unsigned char chip, unsigned char \*hour, unsigned char \*min, unsigned char \*sec, unsigned char \*hsec) — 读出 RTC 的当前时间。

指针 \*hour, \*min, \*sec 和 \*hsec 指向接收小时、分钟、秒和百分之一秒的变量。

例子：

```
#asm
    .equ __i2c_port=0x18
    .equ __sda_bit=3
```

```

.equ __scl_bit=4
#endasm
#include <pcf8583.h>
void main(void) {
  unsigned char h,m,s,hs;
  /* 初始化 I2C 总线 */
  i2c_init();
  /* 初始化 RTC 0, 闹铃只有时间起作用 */
  rtc_init(0,0);
  /* 读 RTC 0 的时间*/
  rtc_get_time(0,&h,&m,&s,&hs);
  /* ..... */
}

```

void rtc\_set\_time(unsigned char chip, unsigned char hour, unsigned char min, unsigned char sec, unsigned char hsec) — 设置 RTC 的时间。

参数 hour, min, sec 和 hsec 对应小时、分钟、秒和百分之一秒的值。

void rtc\_get\_date(unsigned char chip, unsigned char \*date, unsigned char \*month, unsigned \*year) — 读出 RTC 的当前日期。

指针 \*date, \*month, \*year 指向接收日、月、年的变量。

void rtc\_set\_date(unsigned char chip, unsigned char date, unsigned char month, unsigned year) — 设置 RTC 的日期。

void rtc\_alarm\_off(unsigned char chip) — 关闭 RTC 的闹铃。

void rtc\_alarm\_on(unsigned char chip) — 打开 RTC 的闹铃。

void rtc\_get\_alarm\_time(unsigned char chip, unsigned char \*hour, unsigned char \*min, unsigned char \*sec, unsigned char \*hsec) — 读出 RTC 闹铃的时间设置值。

指针 \*hour, \*min, \*sec 和 \*hsec 指向接收小时、分钟、秒和百分之一秒的变量。

void rtc\_set\_alarm\_time(unsigned char chip, unsigned char hour, unsigned char min, unsigned char sec, unsigned char hsec) — 设置 RTC 的闹铃的时间值。

参数 hour, min, sec 和 hsec 对应小时、分钟、秒和百分之一秒的值。

void rtc\_get\_alarm\_date(unsigned char chip, unsigned char \*date, unsigned char \*month) — 读出 RTC 的闹铃的日期设置值。

指针 \*date, \*month, \*year 指向接收日、月、年的变量。

void rtc\_set\_alarm\_date(unsigned char chip, unsigned char date, unsigned char month) — 设置 RTC 的闹铃的日期值。

## 17. Dallas Semiconductor DS1302 Real Time Clock Functions — DS1302 实时时钟函数

只有商业版的 CodeVisionAVR C Compiler 才有这部分功能。

这些函数的原型放在 “.\INC” 目录的 “ds1302.h” 头文件中。使用这些之前必须用 “#include” 包含头文件。

包含头文件之前，你必须先声明那些口线与 DS1302 通讯。

例子：

```

/* DS1302 接在 PORTB */
/* IO — PB 3 */

```

```

/* SCLK — PB 4 */
/* RST — PB 5 */
#asm
    .equ __ds1302_port=0x18
    .equ __ds1302_io=3
    .equ __ds1302_sclk=4
    .equ __ds1302_rst=5
#endasm
/* 包含 DS1302 头文件 */
#include <ds1302.h>

```

void rtc\_init(unsigned char tc\_on, unsigned char diodes, unsigned char res) — 初始化 DS1302。

调用其它的 DS1302 函数之前必须先调用此函数。

如果参数 tc\_on 为 1 则打开涓流充电。

参数 diodes 设定了涓流充电时使用二极管的个数，可以是 1 或 2。

参数 res 指明了涓流充电时的电阻值：

0 — 没有电阻

1 — 2K 欧

2 — 4K 欧

3 — 8K 欧

unsigned char ds1302\_read(unsigned char addr) — 在 DS1302 的 SRAM 中的地址 addr 处读一个字节。

void ds1302\_write(unsigned char addr, unsigned char data) — 在 DS1302 的 SRAM 中的地址 addr 处写一个字节 data。

void rtc\_get\_time(unsigned char \*hour, unsigned char \*min, unsigned char \*sec) — 读出 RTC 的时间。

指针\*hour, \*min, \*sec 指向接收小时、分钟、秒的变量。

例子：

```

#asm
    .equ __ds1302_port=0x18
    .equ __ds1302_io=3
    .equ __ds1302_sclk=4
    .equ __ds1302_rst=5
#endasm
#include <ds1302.h>
void main(void) {
    unsigned char h,m,s;
    /* 初始化 DS1302: 使用涓流充电、一个二极管和 8K 的电阻 */
    rtc_init(1,1,3);
    /* 读 DS1302 时间 */
    rtc_get_time(&h,&m,&s);
    /* ..... */
}

```



`void rtc_set_time(unsigned char hour, unsigned char min, unsigned char sec)` — 设置 RTC 的时间。

参数 `hour`, `min`, `sec` 对应小时、分钟、秒的值。

`void rtc_get_date(unsigned char *date, unsigned char *month, unsigned char *year)` — 读出 RTC 的当前日期。

指针 `*date`, `*month`, `*year` 指向接收日、月、年的变量。

`void rtc_set_date(unsigned char date, unsigned char month, unsigned char year)` — 设置 RTC 的日期。

#### 18. Dallas Semiconductor DS1307 Real Time Clock Functions — DS1307 实时时钟函数

只有商业版的 CodeVisionAVR C Compiler 才有这部分功能。

这些函数的原型放在 “..\INC” 目录的 “ds1302.h” 头文件中。使用这些之前必须用 “#include” 包含头文件。

I2C 总线函数原型在 ds1307.h 中自动包含。

包含头文件之前，你必须先声明那些口线通过 I2C 总线与 PCF8583 通讯。

例子：

```
/* 使用 PORTB 作 I2C 总线 */
```

```
/* SDA 为 PB3 */
```

```
/* SCL 为 PB4 */
```

```
#asm
```

```
    .equ __i2c_port=0x18
```

```
    .equ __sda_bit=3
```

```
    .equ __scl_bit=4
```

```
#endasm
```

```
/* 包含 DS1307 头文件 */
```

```
#include <ds1307.h>
```

`void rtc_init(unsigned char rs, unsigned char sqwe, unsigned char out)` — 初始化 DS1307。

调用这个函数之前，必须调用函数 `i2c_init` 初始化 I2C 总线。

调用其它的 DS1307 函数之前，必须先调用此函数。

参数 `rs` 设定了 SQW/OUT 引脚上输出方波的频率：

0 — 1Hz

1 — 4096Hz

2 — 8192Hz

3 — 32768Hz.

如果参数 `sqwe` 设为 1 则允许 SQW/OUT 引脚上的方波输出。

参数 `out` 设定了禁止 SQW/OUT 引脚上的方波输出时 (`sqwe=0`) 引脚上的逻辑电平。

`void rtc_get_time(unsigned char *hour, unsigned char *min, unsigned char *sec)` — 读出 RTC 的当前时间。

指针 `*hour`, `*min`, `*sec` 指向接收小时、分钟、秒的变量。

例子：

```
/* I2C 总线接在 PORTB */
```

```
/* SDA — PB3 */
```

```

/* SCL — PB4 */
#asm
    .equ __i2c_port=0x18
    .equ __sda_bit=3
    .equ __scl_bit=4
#endasm
#include <ds1307.h>
void main(void) {
    unsigned char h,m,s;
    /* 初始化 I2C 总线 */
    i2c_init();
    /* 初始化 DS1307 */
    rtc_init(0,0,0);
    /* 读 DS1307 时间 */
    rtc_get_time(&h,&m,&s);
    /* ..... */
}

```

void rtc\_set\_time(unsigned char hour, unsigned char min, unsigned char sec) — 设置 RTC 的时间。

参数 hour, min, sec 对应小时、分钟、秒的值。

void rtc\_get\_date(unsigned char \*date, unsigned char \*month, unsigned char \*year) — 读出 RTC 的当前日期。

指针 \*date, \*month, \*year 指向接收日、月、年的变量。

void rtc\_set\_date(unsigned char date, unsigned char month, unsigned char year) — 设置 RTC 的日期。

## 19. 1 Wire Protocol Functions — 单线通讯协议函数

只有商业版的 CodeVisionAVR C Compiler 才有这部分功能。

这些函数的原型放在 “..\INC” 目录的 “1wire.h” 头文件中。使用这些之前必须用 “#include” 包含头文件。

这些函数以 MCU 为主机，外设（单线总线器件）为从机。

包含头文件之前，你必须先声明那些口线使用单线通讯协议与器件通讯。

例子：

```

/* 单线总线在 PORTB */
/* 数据线用 PB2 */
#asm
    .equ __w1_port=0x18
    .equ __w1_bit=2
#endasm
/* 包含头文件 */
#include <1wire.h>

```

由于单线协议函数使用时有严格的延时，所以操作期间要关闭中断。

一定要在 **Project|Configure|C Compiler** 菜单设定正确的晶振频率。

`unsigned char w1_init(void)` — 初始化总线上的器件。

如果有器件返回 1，否则返回 0。

`unsigned char w1_read(void)` — 从总线上读一个字节。

`unsigned char w1_write(unsigned char data)` — 在总线上写一个字节。

如果写过程正常完成返回 1，否则返回 0。

`unsigned char w1_search(unsigned char cmd,void *p)` — 返回总线上器件的个数。

如果没有器件，返回 0。

参数 `cmd` 是发给器件的命令，如 DS1820/DS1822 的搜索 ROM (Search ROM) — F0h，报警搜索 (Alarm Search) — Ech。

指针 `p` 指向存放器件返回的 8 字节 ROM 码的 SRAM 区域。8 字节后会存放某些器件的状态字节，如 DS2405。所以必须给每个器件开辟 9 个字节的 SRAM。

如果总线上有多个器件，那么首先要使用 `w1_search` 读出所有器件的 ROM 码，以便在后面的过程中对它们进行寻址。

例子：

```
#include <90s8515.h>
/* 指定用作单线总线的口和口线 */
#asm
    .equ __w1_port=0x18 ;PORTB
    .equ __w1_bit=2
#endasm
/* 包含单线总线的头文件 */
#include <1wire.h>
/* 包含有 printf 函数原型的头文件 */
#include <stdio.h>
/* 单线总线上器件的最大个数*/
#define MAX_DEVICES 8
/* 定义存放 ROM 码和状态字节的 SRAM 区域 */
unsigned char rom_codes[MAX_DEVICES,9];
/* 晶振频率 Hz */
#define xtal 4000000L
/* 波特率 */
#define baud 9600
void main(void) {
    unsigned char i,j,devices;
    /* 初始化 UART 波特率 */
    UBRR=xtal/16/ baud-1;
    /* 初始化 UART 控制寄存器 */
    UCR=8;
    /* 检测有多少个 DS1820/DS1822，并存放它们的 ROM 码到 rom_codes 数组 */
    devices=w1_search(0xf0,rom_codes);
    /* 显示每一个器件的 ROM 码 */
    printf("%-u DEVICE(S) DETECTED\n\r",devices);
```

```

if (devices) { for (i=0;i<devices;i++) {
    printf("DEVICE #%-u ROM CODE IS:", i+1);
    for (j=0;j<8;j++) printf("%-X ",rom_codes[i,j]);
    printf("\n\r");
    };
};
while (1);
}

```

unsigned char w1\_crc8(void \*p, unsigned char n) — 返回从地址 p 开始的 n 个字节的 8 位 CRC 校验。

## 20. Dallas Semiconductor DS1820/DS1822 Temperature Sensors Functions — DS1820/1822 温度传感器函数

只有商业版的 CodeVisionAVR C Compiler 才有这部分功能。

这些函数的原型放在 “.\INC” 目录的 “ds1820.h” 头文件中。使用这些之前必须用 “#include” 包含头文件。

单线总线的函数原型自动在 ds1820.h 中被包含。

包含头文件之前，你必须先声明那些口线使用单线通讯协议与器件通讯。

例子：

```

/* 指定单线总线使用的口和口线 */
#asm
    .equ __w1_port=0x18 ;PORTB
    .equ __w1_bit=2
#endasm
/* 包含头文件 */
#include <ds1820.h>

```

int ds1820\_temperature\_10(unsigned char \*addr) — 返回 ROM 码存在地址 addr 处的数组中的 DS1820/DS1822 的温度。

温度的值为 0.1 摄氏度。如果有错误，返回值为 -9999。

如果只有 1 个 DS1820/DS1822，就不需要 ROM 码，指针 addr 要设为 NULL (0)。

如果有多个器件要首先读 ROM 码对每一个器件进行识别，然后才能在调用 ds1820\_temperature\_10 时对需要的器件通过 ROM 码进行地址匹配。

例子：

```

#include <90s8515.h>
/* 指定用作单线总线的口和口线 */
#asm
    .equ __w1_port=0x18 ;PORTB
    .equ __w1_bit=2
#endasm
/* 包含单线总线的头文件 */
#include <1wire.h>
/* 包含有 printf 函数原型的头文件 */
#include <stdio.h>

```

```

/*包含有 abc 函数原型的头文件*/
#include <math.h>
/* 单线总线上 DS1820 的最大个数*/
#define MAX_DEVICES 8
/* DS1820/DS1822 的 ROM 码存储区, 每个器件 9 字节, 前 8 字节是 ROM 码, 一个字
节是 CRC */
unsigned char rom_codes[MAX_DEVICES,9];
main()
{unsigned char i,j,devices;int temp;
/* 初始化 UART 波特率 */
UBRR=xtal/16/ baud-1;
/* 初始化 UART 控制寄存器 */
UCR=8;
/* 检测有多少个 DS1820/DS1822, 并存放它们的 ROM 码到 rom_codes 数组 */
devices=w1_search(0xf0,rom_codes);
/* 显示个数 */
printf("%-u DEVICE(S) DETECTED\n\r",devices);
/* 如果没有器件则系统挂起 */
if (devices==0) while (1);
/* 测量并显示温度 */
while (1) { for (i=0;i<devices;)
    {temp=ds1820_temperature_10(&rom_codes[i,0]);
    printf("t%-u=%-i.%-u\x8C\n\r",++i,temp/10,
    abs(temp%10));
    };
};
}

```

unsigned char ds1820\_set\_alarm(unsigned char \*addr,signed char temp\_low,signed char temp\_high) — 设置 DS1820/DS1822 的低温、高温报警温度。

如果设置成功返回 1, 否则返回 0。

报警温度存在 DS1820/DS1822 的暂存器 SRAM 和 EEPROM 中。

用来寻址器件的 ROM 码放在 addr 指向的数组。

如果只有 1 个 DS1820/DS1822, 就不需要 ROM 码, 指针 addr 要设为 NULL (0)。

DS1820/DS1822 的温度报警状态可以用 w1\_search 函数发送报警搜索 (Alarm Search)

— Ech 命令检测到。

例子:

```

#include <90s8515.h>
/* 指定用作单线总线的口和口线 */
#asm
    .equ __w1_port=0x18 ;PORTB
    .equ __w1_bit=2
#endasm
/* 包含单线总线的头文件 */

```

```

#include <ds1820.h>
/* 包含有 printf 函数原型的头文件 */
#include <stdio.h>
/*包含有 abc 函数原型的头文件*/
#include <math.h>
/* 单线总线上 DS1820 的最大个数*/
#define MAX_DEVICES 8
/* DS1820/DS1822 的 ROM 码存储区，每个器件 9 字节，前 8 字节是 ROM 码，一个字
节是 CRC */
unsigned char rom_codes[MAX_DEVICES,9];
/* 给发生温度报警的器件分配 ROM 码存储空间*/
unsigned char alarm_rom_codes[MAX_DEVICES,9];
main()
{unsigned char i,j,devices;
int temp;
/* 初始化 UART 波特率 */
UBRR=xtal/16/baud-1;
/* 初始化 UART 控制寄存器 */
UCR=8;
/* 检测有多少个 DS1820/DS1822，并存放它们的 ROM 码到 rom_codes 数组 */
devices=w1_search(0xf0,rom_codes);
/* 显示个数 */
printf("%-u DEVICE(S) DETECTED\n\r",devices);
/* 如果没有器件则系统挂起 */
if (devices==0) while (1); /* loop forever */
/* 设置所有器件低温报警 25 摄氏度，高温报警 35 摄氏度*/
for (i=0;i<devices;i++)
    {printf("INITIALIZING DEVICE #%-u ", i+1);
    if (ds1820_set_alarm(&rom_codes[i,0],25,35))
        putsf("OK"); else putsf("ERROR");
    };
while (1)
    {/* 测量并显示温度 */
    for (i=0;i<devices;)
        {temp=ds1820_temperature_10(&rom_codes[i,0]);
        printf("t%-u=%-i.%-u\x8C\n\r",++i,temp/10,
        abs(temp%10));
        };
    /* 显示发生温度报警的器件的号码*/
printf("ALARM GENERATED BY %-u DEVICE(S)\n\r",
w1_search(0xec,alarm_rom_codes));
    };
}

```

## 21. SPI Functions — SPI 函数

这些函数的原型放在“..\INC”目录的“spi.h”头文件中。使用这些之前必须用“#include”包含头文件。

`unsigned char spi(unsigned char data)` — 发送一个字节，同时接收一个字节。

调用 `spi` 函数之前要先设置 SPI 控制寄存器 `SPCR`。

`spi` 函数通讯使用查询方式，所以不需要设置 SPI 中断允许标志位 `SPIE`。

下面是一个使用 `spi` 函数与一个 AD7896 的接口示例：

/\* AD 转换使用 AD7896 与 AT90S8515 使用 SPI 总线相连

MCU: AT90S8515

内存模式: SMALL

数据堆栈: 128 字节

晶振频率: 4MHz

AD7896 与 AT90S8515 的连接:

[AD7896] — [AT9S8515 DIP40]

1 Vin

2 Vref=5V

3 AGND — 20 GND

4 SCLK — 8 SCK

5 SDATA — 7 MISO

6 DGND — 20 GND

7 CONVST — 2 PB1

8 BUSY — 1 PB0

2x16 的字符型 LCD 接在 PORTC:

[LCD] — [AT90S8515 DIP40]

1 GND — 20 GND

2 +5V — 40 VCC

3 VLC

4 RS — 21 PC0

5 RD — 22 PC1

6 EN — 23 PC2

11 D4 — 25 PC4

12 D5 — 26 PC5

13 D6 — 27 PC6

14 D7 — 28 PC7 \*/

#asm

.equ \_\_lcd\_port=0x15

#endasm

#include <lcd.h> // 包含 LCD 头文件

#include <spi.h> // 包含 SPI 头文件

#include <90s8515.h>

#include <stdio.h>

#include <delay.h>

// AD7896 参考电压[mV]

```

#define VREF 5000L
// AD7896 控制信号
#define ADC_BUSY PINB.0
#define NCONVST PORTB.1
// LCD 显示缓存
char lcd_buffer[33];
unsigned read_adc(void)
{
    unsigned result;
    // 开启采样模式 1 (高速采样)
    NCONVST=0;
    NCONVST=1;
    // 等待采样完成
    while (ADC_BUSY);
    // 通过 SPI 读 MSB
    result=(unsigned) spi(0)<<8;
    //通过 SPI 读 LSB 并与 MSB 合并
    result|=spi(0);
    // 计算采样电压[mV]
    result=(unsigned) (((unsigned long) result*VREF)/4096L);
    // 返回测量值
    return result;
}

void main(void)
{
    // 初始化 PORTB
    // PB.0 输入, 接 AD7896 忙信号 (BUSY)
    // PB.1 输出, 接 AD7896 启动采样 (/CONVST)
    // PB.2 , PB.3 输入
    // PB.4 输出 (SPI /SS)
    // PB.5 输入
    // PB.6 输入 (SPI MISO)
    // PB.7 输出 (SPI SCLK)
    DDRB=0x92;
    // 初始化 SPI 在主机模式
    // 不需要中断, MSB 先发送, 时钟极性负, SCK 空闲时为低
    // SCK=fxtal/4
    SPCR=0x54;
    // AD7896 工作在模式 1 (高速采样)
    // /CONVST=1, SCLK=0
    PORTB=2;
    // 初始化 LCD
    lcd_init(16);
    lcd_putsf("AD7896 SPI bus\nVoltmeter");
    delay_ms(2000);
}

```



```

lcd_clear();
// 读并显示 ADC 输入电压
while (1)
{
    sprintf(lcd_buffer,"Uadc=%4umV",read_adc());
    lcd_clear();
    lcd_puts(lcd_buffer);
    delay_ms(100);
};
}

```

## 22. Power Management Functions — 电源管理函数

这些函数的原型放在“..\INC”目录的“sleep.h”头文件中。使用这些之前必须用“#include”包含头文件。

`void sleep_enable(void)` — 允许进入低功耗模式。

`void sleep_disable(void)` — 禁止进入低功耗模式。

这可以防止意外进入低功耗模式。

`void idle(void)` — 进入空闲模式。

调用这个函数之前必须先调用 `sleep_enable` 函数，以允许进入低功耗模式。在这种模式下，CPU 停止工作，但定时器、看门狗和中断系统继续工作。MCU 可以由使能的中断唤醒。

`void powerdown(void)` — 进入掉电模式。

调用这个函数之前必须先调用 `sleep_enable` 函数，以允许进入低功耗模式。在这种模式下，外部晶振停振，看门狗和外部中断继续工作。只有外部复位、看门狗复位和外部中断可以唤醒 MCU。

`void powersave(void)` — 进入省电模式

调用这个函数之前必须先调用 `sleep_enable` 函数，以允许进入低功耗模式。这种模式与掉电模式类似，但略有不同，参考 Atmel 的数据手册。

## 23. Gray Code Conversion Functions — 格雷码转换函数

这些函数的原型放在“..\INC”目录的“gray.h”头文件中。使用这些之前必须用“#include”包含头文件。

`unsigned char gray2binc(unsigned char n)`

`unsigned char gray2bin(unsigned int n)`

`unsigned char gray2binl(unsigned long n)`

— 将格雷码形式的 `n` 转换到二进制形式。

`unsigned char bin2grayc(unsigned char n)`

`unsigned char bin2gray(unsigned int n)`

`unsigned char bin2grayl(unsigned long n)`

— 将二进制形式的 `n` 转换到格雷码形式。