查缺补漏笔记

目录

[查缺补漏笔记 1](#_Toc50627920)

[看门狗 2](#_Toc50627921)

[BootLoader 2](#_Toc50627922)

[嵌入式Linux系统 2](#_Toc50627923)

[常用关键字 3](#_Toc50627924)

[#ifndef用法 3](#_Toc50627925)

[#if和if 4](#_Toc50627926)

[EEPROM 4](#_Toc50627927)

[FLASH 4](#_Toc50627928)

[闪存和内存的差异 4](#_Toc50627929)

[指针数组和数组指针 4](#_Toc50627930)

[开环和闭环 5](#_Toc50627931)

[位段 5](#_Toc50627932)

[RAM使用 5](#_Toc50627933)

[KEIL查看单片机内存 5](#_Toc50627934)

[回调函数 6](#_Toc50627935)

[位操作用法：掩码(mask) 6](#_Toc50627936)

[文件流 7](#_Toc50627937)

[①FILE\* fopen(char const \*\_name, char const \*\_mode);不安全 7](#_Toc50627938)

[②errno\_t fopen\_s(FILE \*\*\_stream, char const \*\_name, char const \*\_mode);安全 7](#_Toc50627939)

[fprintf\_s(FILE \*\_stream,const char\* \_format,…) 8](#_Toc50627940)

[fscanf\_s(FILE \*\_stream,const char\* \_format,…) 8](#_Toc50627941)

[fputs(const char \*\_buffer,FILE \*\_stream) 8](#_Toc50627942)

[fgets(char \*\_buffer,int \_max\_count,FILE \*\_stream) 8](#_Toc50627943)

[fseek(FILE \*\_stream,long \_offset, int \_fromwhere) 9](#_Toc50627944)

# 看门狗

看门狗定时器（WDT、WDOG）：在发生软件故障时，通过使器件复位（如果软件未将器件清零）将单片机复位。也可以用于将器件从休眠或空闲模式唤醒，看门狗定时器对微控制器提供了独立的保护系统，当系统出现故障时，在可选的超市周期之后，看门狗以RESET信号作出响应，当程序死机时，就会使单片机复位。

它实际上是一个计数器，一般给看门狗一个数字，程序开始运行后看门狗开始计数，如果程序运行正常，过一段时间CPU应发出指令让看门狗清零，重新开始计数。如果看门狗增加到设定值就认为程序没有正常工作，强制整个系统复位。

喂狗：看门狗计数器设定的时间内没有溢出并告诉他重新计数，把计数器清零。

# BootLoader

启动装载（BootLoader）：在嵌入式操作系统中，BootLoader时在系统内核运行之前运行。系统上电或复位通常都从地址0x00000000处开始执行，而在这个地址处安排的通常就是系统的BootLoader程序。

# 嵌入式Linux系统

从软件的角度看通常可以分为四个层次：

1. 引导加载程序：包含固话在固件（firmware）中的boot代码（可选）和BootLoader两大部分。
2. Linux内核：特定于嵌入式板子的定制内核以及内核的启动参数。
3. 文件系统：包括根文件系统和建立于Flash内存设备之上文件系统。通常用ramdisk来作为rootfs。
4. 用户应用程序：特定于用户的应用程序，有时在用户应用程序和内核层之间可能还会包括一个嵌入式图形用户界面。

# 常用关键字

const：只读变量 1、char \*const cp; 指向char的const指针

2、char const \*pc1; 指向const char指针

3、const char \*pc2; 指向const char指针

4、const char \*const pc3; 指向const char的const指针

cp指向地址是常量，不能修改，但可以修改指针值；

pc1、pc2指向值为常量，不能修改，但可以修改指向地址；

pc3指向地址和指向值都是常量，不能修改

|  |  |  |
| --- | --- | --- |
|  | 指向地址 | 指向值 |
| cp | 不可变 | 可变 |
| pc1、pc2 | 可变 | 不可变 |
| pc3 | 不可变 | 不可变 |

extern：声明变量实在其他文件中声明（也可以看做是引用变量）

static：声明静态变量

volatile：说明变量在程序执行中可被隐含地改变（指出变量随时可能发生变化，与volatile变量有关的运算，不要进行编译优化，以免出错）

define：宏定义 #define uchar unsigned char （不进行检测）

typedef：声明类型 typedef unsigned char uchar;（进行数据类型检测）

# #ifndef用法

#ifndef都是一种宏定义判断，作用是防止多重定义。

#ifndef是if not define的简写。

一般的使用场景为：

1）、头文件中使用，防止头文件被多重调用

2）、作为测试使用，省去注释代码的麻烦

3）、作为不同角色或者场景的判断使用。

头件的中的#ifndef，这是一个很关键的东西。比如你有两个C文件，这两个C文件都include了同一个头文件。而编译时，这两个C文件要一同编译成一个可运行文件，于是问题来了，大量的声明冲突。

还是把头文件的内容都放在#ifndef和#endif中吧。不管你的头文件会不会被多个文件引用，你都要加上这个。一般格式是这样的：

#ifndef <标识>

#define <标识> ......

#endif

# #if和if

#if和if都是C和C++中的指令，区别如下：

1、属性作用不同

#if是条件编译，条件编译是根据宏条件选择性地编译语句；而if是条件语句，条件语句是根据条件表达式选择性地执行语句。

2、执行时期不同

#if是编译器在编译代码时完成的，if是在程序运行时进行的在程序运行时执行。

# EEPROM

定义（Electrically Erasable Programmable read only memory）：带电可擦可编程只读存储器，一种掉电后数据不丢失的存储芯片。

用途：存储更新数据

其他用法：为降低成本，可以使用FLASH代替

# FLASH

FLASH闪存（Flash Memory）：内存器件，不挥发性（Non-Volatile）内存

# 闪存和内存的差异

挥发性内存（DDR、SDRAM、RDRAM）：只要停止电流供应，内存中的数据便无法保持，因此每次电脑开机都需要把数据重新载入内存

不挥发性内存（Flash）：在没有电流供应的条件下也能够长久地保持数据，其存储特性相当于硬盘

# 指针数组和数组指针

指针数组：首先它是一个数组，数组的元素都是指针，数组占多少字节由数组本身的大小决定，每一个元素都是一个指针，在32位系统下任何类型的指针永远都是4个字节，它是“存储指针的数组”的简称。 例：int \*p1[10];

数组指针：首先它是一个指针，它指向一个数组。在32位系统下任何类型的指针永远都是占4个字节，至于它指向的数组占多少字节，具体要看数组大小，它是“指向数组的指针”的简称。 例：int (\*p2)[10];

# 开环和闭环

开环：不将控制的结果反馈回来影响当前控制的系统

闭环：反馈控制系统，是将系统输出量的测量值与所期望的给定值相比较，由此产生一个偏差信号，利用此偏差信号进行调节控制，使输出值尽量接近于期望值。

# 位段

定义：c语言特有的数据结构，允许我们定义一个由位组成的段，并可以为它赋以一个名字。（超出数据长度的数据将被自动截掉）

# RAM使用

rwdata：代表已初始化的读写数据，程序中定义并且初始化的全局变量和静态变量位于此处。

zidata：代表未初始化的读写数据，程序中定义但没有初始化的全局变量和静态变量位于此处。

#pragma arm section rwdata = "RAM\_KEEP"

#pragma arm section zidata = "RAM\_KEEP" //以下变量放入RAM\_KEEP

CTRL\_t CTRL;

Vender\_t VenderSysMode;

Uchar WorkState;

#pragma arm section rwdata //以下变量恢复默认

#pragma arm section zidata

# KEIL查看单片机内存

Code：代码指令占用的空间

RO-data（Read Only Data）：只读常量占用的空间，比如const型常量、字符串常量等。

RW-data（Read Write Data）：可读可写并且已初始化了的变量所占用的空间，比如全局变量、静态变量等。

ZI-data（Zero Initialize Data）：以0初始化的变量，比如未初始化赋值的全局变量、静态变量等。

烧写程序：FLASH占用空间大小=Code + RO-data + RW-data。

运行RAM空间大小 = RW-data + ZI-data + Stack\_Size + Heap\_Size。

# 回调函数

定义：

指针是一个变量，是用来指向内存地址的。

一个程序运行时，所有和运行相关的物件都是需要加载到内存中，这就决定了程序运行时的任何物件都可以用指针来指向它。

函数是存放在内存代码区域内的，它们同样有地址，因此同样可以用指针来存取函数，把这种指向函数入口地址的指针称为函数指针。

# 位操作用法：掩码(mask)

定义：一些设置开（1）和关（0）的位组合。

打开位（设置位）：使用按位或（|）运算

关闭位（清空位）：使用按位与（&）取反（~）运算

flag = 0000 1111,mask = 0000 0010

flag &= ~mask

切换位：使用按位按位异或（^）运算，打开已关闭的位，或关闭已打开的位

检查位的值：使用条件if((flag & mask) == mask);

乘法除法用移位操作，效率更高，在乘以或除以2n的整数时都可以使用左右移位来提高效率，若是其他整数，也可用移位操作。

相应2的乘法公式为2\*num = (2 << (接近乘数的2的n次方的n)) + (num - 接近乘数的2的n次方的n) <<1

例： 2 \* 9 = 18 BIN: 0001 0010

2 << 3 + 1<<1 BIN: 0000 0010 << 3 +1<<1 = 0001 0000 + 0010 = 0001 0010

求余运算用移位操作，效率也高

相应的求余公式为x = num &(2的n次方-1)

例：5 % 2 = 1 5 & 1 = 1

# 文件流

## ①FILE\* fopen(char const \*\_name, char const \*\_mode);不安全

## ②errno\_t fopen\_s(FILE \*\*\_stream, char const \*\_name, char const \*\_mode);安全

①返回值：成功文件指针，失败返回NULL

②返回值：成功0，失败非0

typedef int errno\_t;

\_stream：一个指向FILE指针的指针

\_name：希望打开的文件或者设备的名字

\_mode：参数提示流是用于只读、只写还是既读又写，以及它是文本流还是二进制流

|  |  |  |  |
| --- | --- | --- | --- |
|  | 读取 | 写入 | 添加 |
| 文本 | “r” | “w” | “a” |
| 二进制 | “rb” | “wb” | “ab” |

注意：1、如果一个文件打开是用于读取，那么它必须原先已经存在。

2、如果一个文件打开是用于写入，如果它原先已经存在，那么它原来的内容就会被删除 如果它原先不存在，那么久创建一个新文件

3、如果一个打开用于添加的文件原先并不存在，那么它将被创建

如果原先存在，它原先的内容并不会被删除

无论在哪一种情况下， 数据只能从文件的尾部写入。

|  |  |  |  |
| --- | --- | --- | --- |
| 文件模式 | 文件原先状态 | 使用效果 | 写入位置 |
| 读取 | 必须存在 |  | 文件尾部 |
| 写入 | 存在 | 内容被删除 | 文件尾部 |
| 不存在 | 创建新文件 | 文件尾部 |
| 添加 | 存在 | 原先的内容不会删除 | 文件尾部 |
| 不存在 | 创建新文件 | 文件尾部 |

## fprintf\_s(FILE \*\_stream,const char\* \_format,…)

用法：将数据写入文本

\_stream：一个指向FILE的指针

const char\* \_format：类似printf用法

例：FILE\* sInput;

errno\_t sEr;

sEr = fopen\_s(&sInput, "data3.txt", "w");

fprintf\_s(sInput,"%s", "Faceless");//写入文本流

## fscanf\_s(FILE \*\_stream,const char\* \_format,…)

用法：将文本数据读取出来，可以通过判断是否等于EOF读取所有数据

\_stream：一个指向FILE的指针

const char\* \_format：类似scanf用法，最好加一下类型长度，类型长度不能少于读取得数据具体长度

例：FILE\* sInput;

errno\_t sEr;

char sStr[100] = { 0 };

sEr = fopen\_s(&sInput, "data3.txt", "r");

fscanf\_s(sInput, "%s", sStr,sizeof(sStr));

## fputs(const char \*\_buffer,FILE \*\_stream)

用法：将数据写入文本

\_buffer：缓冲区输入

\_stream：一个指向FILE的指针

例：FILE\* sInput;

errno\_t sEr;

sEr = fopen\_s(&sInput, "data3.txt", "w");

fputs("Faceless", sInput);

## fgets(char \*\_buffer,int \_max\_count,FILE \*\_stream)

用法：将文本数据读取出来，可以通过判断是否等于NULL读取所有数据，包括换行符（区别fscanf\_s）

\_buffer：缓冲区存储

\_max\_count：读取的最大内存空间

\_stream：一个指向FILE的指针

例：FILE\* sInput;

errno\_t sEr;

char sStr[100] = { 0 };

sEr = fopen\_s(&sInput, "data3.txt", "r");

while (fgets(sStr, 100, sInput) != NULL)//文本数据读取

{

printf("%s", sStr);

}

fclose(sInput);

## fseek(FILE \*\_stream,long \_offset, int \_fromwhere)

\_stream：一个指向FILE的指针

\_offset：指针偏移量

\_fromwhere：偏移起始位 文件0（SEET\_SET）

当前位置1（SEET\_CUR）

文件尾2（SEET\_END）

用法：函数设置文件指针stream的位置

如果执行成功，stream将指向以fromwhere为基准，偏移offset个字节的位置。如果执行失败（比如offset超过文件自身大小），则不改变stream指向的位置。