第三章 软硬件规范以及 STM32 最小系统

本章开始，我们正式进入“乐创 DIY”的“STM32 单片机实战教程”。由于我们是一个社区类的教程视频，因此我们必须统一我们的软件和硬件规则，以保证所在这个社群的人都能看懂我们的代码和硬件。所以，我觉得把规范作为我们设计课程中的正式第一课是十分有必要的。

1 软件规范

在这节课正式之前，我们先来比较一下图 1 中的两份代码片段。

```
void UART1_receive(void) interrupt 16
{xdata uint num=0;
xdata uint  loop_wifi=0;
EA=0;EIE2=EIE2&0XFD;//ES1=0;//关串口中断
for(loop_wifi=0;loop_wifi<25;loop_wifi++){receivex[loop_wifi]=0;}
for(loop_wifi=0;loop_wifi<25;loop_wifi++){
while((SCON1&0X01)==0)
{num++;if(num>0x3200)break;}//
receivex[loop_wifi]=SBUF1;
SCON1=0X70;num=0;
}
EIE2=EIE2|0X02;//ES1=1;//开串口中断
EA=1;
}
```

```
void UART1_receive(void) interrupt 16
{
    xdata uint  num=0;
    xdata uint  loop_wifi=0;
    EA=0;
    EIE2=EIE2&0XFD;
    //ES1=0;//关串口中断
    for(loop_wifi=0;loop_wifi<25;loop_wifi++)
        receivex[loop_wifi]=0;
    for(loop_wifi=0;loop_wifi<25;loop_wifi++)
    {
        while((SCON1&0X01)==0)
        {
            num++;
            if(num>0x3200)
                break;
        }
        receivex[loop_wifi]=SBUF1;
        SCON1=0X70;num=0;
    }
    EIE2=EIE2|0X02;//ES1=1;//开串口中断
    EA=1;
}
```

图 1 代码风格比对

相信绝大多数朋友喜欢右边这张图片里面的代码编写方式，尽管在变量名称定义上面还是没有做的很好，但就条理这个层面来说，右边的代码风格条理清晰，层次分明，让人一看就能知道每个 if，for 所作用的范围。其实，一种好的代码风格，不仅让人更易读，以便后人维护，还能在写程序的时候，让自己能够逻辑清晰，不会有错。但是就这种最简单，只要墨守成规就能做好的事情，竟然没有一个学校老师，会花时间在课堂上面讲，以至于当今很多所谓的“精通 C 语言工程师”，写出了左图中的代码。

以下，我们就开始软件规范的讲解，我们下文中的内容，适当参考了华为软件编程规范和范例。以后的每节课，我们都会严格按照今天提到的风格来写我们的代码。

1.1 排版

（1）程序块要采用缩进风格编写，缩进的空格数为 4 个。函数或过程的开始、结构的定义及循环、判断等语句中的代码都要采用缩进风格，case 语句下的情况处理语句也要遵从语句缩进要求。

说明：由于每个 IDE 的文本编辑器自动缩进的空格数可能不一样，因此建议缩进时，手动敲击 4 个空格按键。

（2）相对独立的程序块之间、变量说明之后，必须加空行。

示例：如下例子不符合规范。

```
if (!valid_ni(ni))
{
    ... // program code
}
repssn_ind = ssn_data[index].repssn_index;
repssn_ni  = ssn_data[index].ni;
```

应如下书写：

```
if (!valid_ni(ni))
{
    ... // program code
}

repssn_ind = ssn_data[index].repssn_index;
repssn_ni  = ssn_data[index].ni;
```

（3）较长的语句（如循环、判断等语句或者函数等）(>80 字符)要分成多行书写，长表达式要在低优先级操作符处划分新行，操作符放在新行之首，划分出的新行要进行适当的缩进，使排版整齐，语句可读。

示例：

```
perm_count_msg.head.len = NO7_TO_STAT_PERM_COUNT_LEN
                          + STAT_SIZE_PER_FRAM * sizeof( _UL );
for (i = 0, j = 0; (i < BufferKeyword[word_index].word_length)
                   && (j < NewKeyword.word_length); i++, j++)
n7stat_flash_act_duration( stat_item, frame_id *STAT_TASK_CHECK_NUMBER
                                    + index, stat_object );
```

（4）不允许把多个短语句写在一行中，即一行只写一条语句。

示例：如下例子不符合规范。

```
rect.length = 0;  rect.width = 0;
```

应如下书写

```
rect.length = 0;
rect.width  = 0;
```

（5）if、for、do、while、case、switch、default 等语句自占一行，且 if、for、do、while 等语句的执行语句部分无论多少都要加括号{}。

示例：如下例子不符合规范。

```
if (pUserCR == NULL) return;
```

应如下书写：

```
if (pUserCR == NULL)
{
    return;
}
```

（6）程序块的分界符(如 C/C++ 语言的大括号‘{’ 和‘}’)应各独占一行并且位于同一列，同时与引用它们的语句左对齐。在函数体的开始、类的定义、结构的定义、枚举的定义以及 if、for、do、while、switch、case 语句中的程序都要采用如上的缩进方式。

示例：如下例子不符合规范。

```
for (...) {
    ... // program code
}

if (...)
    {
    ... // program code
    }

void example_fun( void )
    {
    ... // program code
    }
```

应如下书写。

```
for (...)
```

```
{
    ... // program code
}

if (...)
{
    ... // program code
}


void example_fun( void )
{
    ... // program code
}
```

（7）在两个以上的关键字、变量、常量进行对等操作时，它们之间的操作符之前、之后或者前后要加空格；进行非对等操作时，如果是关系密切的立即操作符(如->)，后不应加空格。

说明：采用这种松散方式编写代码的目的是使代码更加清晰。

由于留空格所产生的清晰性是相对的，所以，在已经非常清晰的语句中没有必要再留空格，如果语句已足够清晰则括号内侧(即左括号后面和右括号前面)不需要加空格，多重括号间不必加空格，因为在 C/C++语言中括号已经是最清晰的标志了。

在长语句中，如果需要加的空格非常多，那么应该保持整体清晰，而在局部不加空格。给操作符留空格时不要连续留两个以上空格。

示例：

(1) 逗号、分号只在后面加空格。

```
int a, b, c;
```

(2) 比较操作符，赋值操作符“=”、“+=”，算术操作符“+”、“%”，逻辑操作符“&&”、“&”，位域操作符“<<”、“^”等双目操作符的前后加空格。

```
if (current_time >= MAX_TIME_VALUE)
a = b + c;
a *= 2;
a = b ^ 2;
```

(3) "!"、"~"、"++"、"--"、"&"(地址运算符)等单目操作符前后不加空格。

```
*p = 'a';          // 内容操作"*"与内容之间
flag = !isEmpty; // 非操作"!"与内容之间
p = &mem;          // 地址操作"&" 与内容之间
i++;               // "++", "--"与内容之间
```

(4) "->"、"."前后不加空格。

```
p->id = pid;      // "->"指针前后不加空格
```

(5) if、for、while、switch 等与后面的括号间应加空格，使 if 等关键字更为突出、明显。

```
if (a >= b && c > d)
```

1.2 注释

（1）一般情况下，源程序有效注释量必须在 20%以上。

说明：注释的原则是有助于对程序的阅读理解，在该加的地方都加了，注释不宜太多也不能太少，注释语言必须准确、易懂、简洁。

（2）说明性文件(如头文件.h 文件、.inc 文件、.def 文件、编译说明文件.cfg 等)头部应进行注释，注释必须列出：版权说明、版本号、生成日期、作者、内容、功能、与其它文件的关系、修改日志等，头文件的注释中还应有函数功能简要说明。

示例：下面这段头文件的头注释比较标准，当然，并不局限于此格式，但上述信息建议要包含在内。

```
/************************************************
  Copyright (C), 1988-1999, Huawei Tech. Co., Ltd.
  File name:      // 文件名
  Author:         Version:          Date:          // 作者、版本及完成日期
  Description:    // 用于详细说明此程序文件完成的主要功能，与其他模块
                  // 或函数的接口，输出值、取值范围、含义及参数间的控
                  // 制、顺序、独立或依赖等关系
  Others:         // 其它内容的说明
  Function List:  // 主要函数列表，每条记录应包括函数名及功能简要说明
    1. ...
  History:        // 修改历史记录列表，每条修改记录应包括修改日期、修改
                  // 者及修改内容简述
    1. Date:
       Author:
       Modification:
    2. ...
************************************************/
```

（3）函数头部应进行注释，列出：函数的目的/ 功能、输入参数、输出参数、返回值、调用关系(函数、表)等

示例：下面这段函数的注释比较标准，当然，并不局限于此格式，但上述

信息建议要包含在内。

```
/*************************************************
  Function:       // 函数名称
  Description:    // 函数功能、性能等的描述
  Calls:          // 被本函数调用的函数清单
  Called By:      // 调用本函数的函数清单
  Table Accessed: // 被访问的表(此项仅对于牵扯到数据库操作的程序)
  Table Updated:  // 被修改的表(此项仅对于牵扯到数据库操作的程序)
  Input:          // 输入参数说明，包括每个参数的作
                  // 用、取值说明及参数间关系。
  Output:         // 对输出参数的说明。
  Return:         // 函数返回值的说明
  Others:         // 其它说明
*************************************************/
```

（4）边写代码边注释，修改代码同时修改相应的注释，以保证注释与代码的一致性。不再有用的注释要删除。

（5）将注释与其上面的代码用空行隔开。

示例：如下例子，显得代码过于紧凑。

```
/* code one comments */
program code one
/* code two comments */
program code two
```

应如下书写

```
/* code one comments */
program code one

/* code two comments */
program code two
```

（6）对变量的定义和分支语句(条件分支、循环语句等)必须编写注释。

说明：这些语句往往是程序实现某一特定功能的关键，对于维护人员来说，良好的注释帮助更好的理解程序，有时甚至优于看设计文档。

（7）对于 switch 语句下的 case 语句，如果因为特殊情况需要处理完一个 case 后进入下一个 case 处理，必须在该 case 语句处理完、下一个 case 语句前加上明确的注释。

1.3 标识符命名

（1）标识符的命名要清晰、明了，有明确含义，同时使用完整的单词或大家基本可以理解的缩写，避免使人产生误解。

说明：较短的单词可通过去掉“元音”形成缩写；较长的单词可取单词的头几个字母形成缩写；一些单词有大家公认的缩写。

示例：如下单词的缩写能够被大家基本认可。

temp 可缩写为 tmp； 临时

flag 可缩写为 flg； 标志

statistic 可缩写为 stat ； 统计

increment 可缩写为 inc； 增量

message 可缩写为 msg； 消息

（2）自己特有的命名风格，要自始至终保持一致，不可来回变化。，关于变量命名没有固定的方式，但是一般分成 Linux 的小写加下划线命名以及 Windows 底下的匈牙利命名法。命名规范必须与所使用的系统风格保持一致，并在同一项目中统一，比如采用 UNIX 的全小写加下划线的风格或大小写混排的方式，不要使用大小写与下划线混排的方式，用作特殊标识如标识成员变量或全局变量的 m_ 和 g_ ，其后加上大小写混排的方式是允许的。

（3）对于变量命名，禁止取单个字符(如 i、j、k...)，建议除了要有具体含义外，还能表明其变量类型、数据类型等，但 i、j、k 作局部循环变量是允许的。

说明：变量，尤其是局部变量，如果用单个字符表示，很容易敲错(如 i 写成 j)，而编译时又检查不出来，有可能为了这个小小的错误而花费大量的查错时间。

示例：下面所示的局部变量名的定义方法可以借鉴。

```
int liv_Width
```

其变量名解释如下：

```
l       局部变量(Local)  (其它：g 全局变量(Global)...)

i       数据类型(Interger)

v       变量(Variable)   (其它：c 常量(Const)...)

Width   变量含义
```

这样可以防止局部变量与全局变量重名。

1.4 可读性

避免使用不易理解的数字，用有意义的标识来替代。涉及物理状态或者含有物理意义的常量，不应直接使用数字，必须用有意义的枚举或宏来代替。

示例：如下的程序可读性差。

```
if (Trunk[index].trunk_state == 0)
```

```
{
    Trunk[index].trunk_state = 1;
    ...  // program code
}
```

应改为如下形式。

```
#define TRUNK_IDLE 0
#define TRUNK_BUSY 1

if (Trunk[index].trunk_state == TRUNK_IDLE)
{
    Trunk[index].trunk_state = TRUNK_BUSY;
    ...  // program code
}
```

1.5 变量、结构

（1） 尽可能少的使用全局变量

说明：首先，公共变量是增大模块间耦合的原因之一；其次，大多数编译器都是把全局变量做成一种静态式的内存分配的，因此太多的全局变量定义，对于 RAM 资源本身就有限的单片机来说，无非是一个比较重大的开销；最后，太多的全局变量，对程序的阅读和维护也会造成严重干扰，长时间的程序维护可能会 BUG 频发。故应减少没必要的公共变量以降低模块间的耦合度。

（2）防止局部变量与公共变量同名。

说明：若使用了较好的命名规则，那么此问题可自动消除。

（3）严禁使用未经初始化的变量作为右值。

说明：特别是在 C/C++中引用未经赋值的指针，经常会引起系统崩溃。

1.6 函数

（1）对所调用函数的错误返回码要仔细、全面地处理

（2） 明确函数功能，精确(而不是近似)地实现函数设计

（3） 编写可重入函数时，应注意局部变量的使用(如编写 C/C++ 语言的可重入函数时，应使用 auto 即缺省态局部变量或寄存器变量)

说明：编写 C/C++语言的可重入函数时，不应使用 static 局部变量，否则必须经过特殊处理，才能使函数具有可重入性。

（3）防止将函数的参数作为工作变量

说明：将函数的参数作为工作变量，有可能错误地改变参数内容，所以很危险。对必须改变的参数，最好先用局部变量代之，最后再将该局部变量的内容赋给该参数。

示例：下函数的实现不太好。

```
void sum_data( unsigned int num, int *data, int *sum )
{
    unsigned int count;
    *sum = 0;

    for (count = 0; count < num; count++)
    {
        *sum  += data[count]; // sum 成了工作变量，不太好。
    }
}
```

若改为如下，则更好些。

```
void sum_data( unsigned int num, int *data, int *sum )
{
    unsigned int count ;
    int sum_temp;
    sum_temp = 0;

    for (count = 0; count < num; count ++)
    {
        sum_temp  += data[count];
    }
```

```
    *sum = sum_temp;
}
```

（4）函数的规模尽量限制在 200 行以内（不包括注释和空格行），一个函数仅完成一件功能。

说明：每个子函数，最好精简，并且功能专一性，最忌讳的就是出现一个功能繁多的庞然大物，这样的代码，在易读性和维护性上面，都是属于比较差的存在。再简单的功能，需要写函数时，哪怕只有一行，也还是不要贪图方便，就不写。

（5）使用动宾词组为执行某操作的函数命名。

示例：参照如下方式命名函数。

```
void print_record( unsigned int rec_ind ) ;
int  input_record( void ) ;
unsigned char get_current_color( void ) ;
```

1.7 代码质量保证优先原则

(1)正确性，指程序要实现设计要求的功能。

(2)稳定性、安全性，指程序稳定、可靠、安全。

(3)可测试性，指程序要具有良好的可测试性。

(4)规范/可读性，指程序书写风格、命名规则等要符合规范。

(5)全局效率，指软件系统的整体效率。

(6)局部效率，指某个模块/子模块/函数的本身效率。

(7)个人表达方式/个人方便性，指个人编程习惯。

总之，一份优秀的代码，不是一节课可以塑造出来的，只有大家在平时的使用中，不断地遵守和注意，才能练成一份漂亮的代码，一个大牛程序员的塑造，并不是仅仅是程序可以完成功能就好，这么简单。再有，推荐大家平时可以研究研究 Linux 源码，因为就 Linux 内核来说，由于它是以网络共享式去完成的，因此它的代码风格简洁，明了，充分体现了我上文中提到的“高内聚，低耦合”这一原则，非常具有借鉴意义。

2 STM32F103C8T6 最小系统

学过我们 51 单片机的同学可能都知道，尽管一片单片机内部已经集成了很多高大上的东西，但是想让它跑起来，还是必须外加一些辅助的器件，我们把 MCU 和这些必备外接器件组合起来的电路，就叫做单片机的最小系统。STM32 也不例外。我们先来仔细看下 STM32F103C8T6 的引脚图以及端口分布（见 STM32F103C8T6 的数据手册）。如图 1 所示，具体的 Pin Map（管脚分配）我们看数据手册。

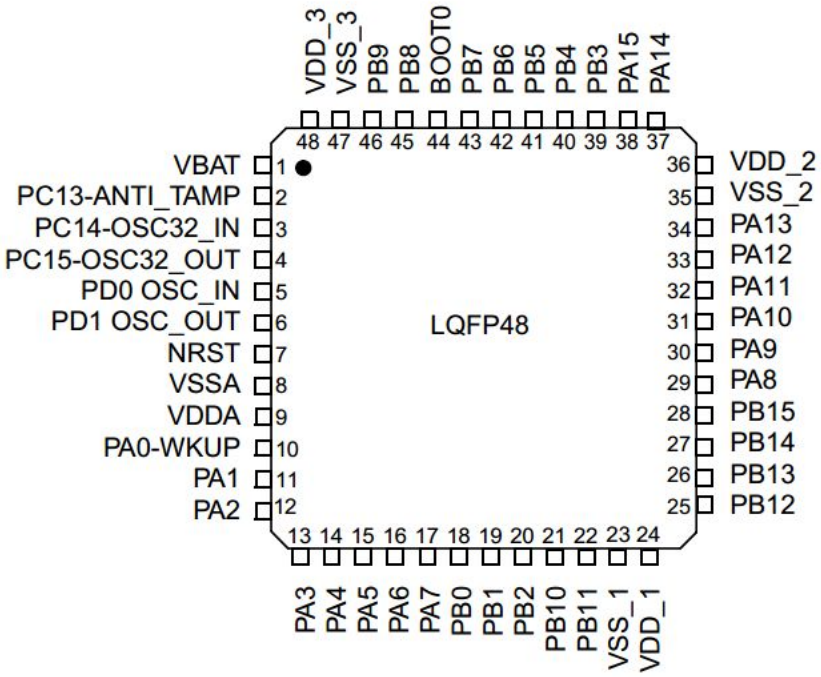

图 2 STM32F103C8T6 引脚图

在这里，主要说明以下几个引脚的内容，如表 1 所示。

表 1. STM32F103C8T6 部分功能引脚图

管脚号	名称	功能
1	VBAT	外部不掉电电池正端接口
3	PC14/OSC32_IN	外接 RTC 晶振（32.768kHz）
4	PC15/OSC32_OUT	外接 RTC 晶振（32.768kHz）
5	OSC_IN	外接系统晶振（一般接 8MHz）
6	OSC_OUT	外接系统晶振（一般接 8MHz）
7	NRST	外部复位引脚，低电平复位
23，35，47	VSS	STM32 系统电压负端（0V）
24，36，48	VDD	STM32 系统电压正端（接 3.3V）
8	VSS_A	STM32 内置 ADC 供电电压负端（0V）
9	VDD_A	STM32 内置 ADC 供电电压（接 3.3V）
20	PB2/BOOT1	系统启动配置引脚，见后文
44	BOOT0	系统启动配置引脚，见后文

上表中，我们还是遗留下来了一个问题，就是关于 BOOT0 和 BOOT1 的配置问题，关于 BOOT0 和 BOOT1 的配置，如表 2 所示。

表 2. STM32F103C8T6 启动设置

BOOT1	BOOT0	启动模式	说明
X	0	主闪存存储器	主闪存存储器被选作启动区
0	1	系统存储器	系统存储器被选作启动区
1	1	内嵌 SRAM	内嵌 SRAM 被选作启动区

接下来，我们结合“STM32F10xxx 硬件开发”文档，设计出了 STM32 的最小系统板，上面包含了复位电路，晶振电路，去耦电容等等。最小系统如图 2 所示。

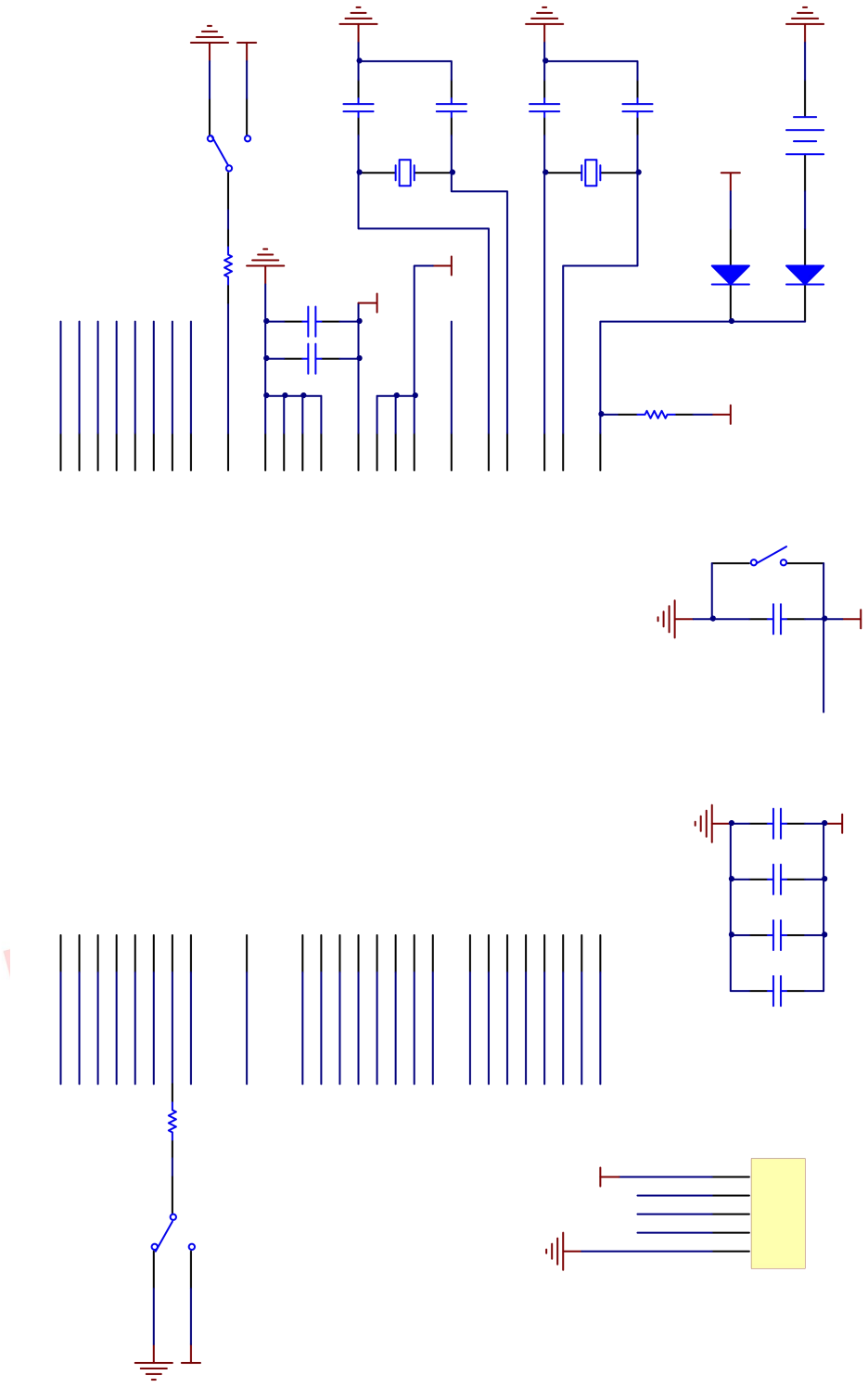

图 2 STM32F103C8T6 最小系统原理图

微信扫码关注“乐创客”