一、 Keil 工程的建立及相关规范

雨帆电子

在学习单片机编程的初期，养成良好的编程习惯是非常必要的，包括从项目的建立、规程规范、程序优化等。这些都可以在前期掌握，因为没有太高的水平要求，只涉及一些技巧和习惯。同时先声明以下只是个人习惯，只做为参考使用。

1.项目的建立

先从项目建立开始，以 **Keil2** 建立项目为例。

因为对于同一项目来讲，开发过程中可能会遇到和解决不同的问题，需要做不小的改动，而改动时某些方面是不可预知的，可能会把之前的功能破坏掉，这时我习惯做一个备份，写上不同的版本号（视程序而定，改动不大就不必要写多个版本了），并附一简单的说明文档，说明完成哪些功能，做了哪些修改，保证在日后自己和他人都能能方便快速地了解该程序。注意最好注明时间，这对之后查阅有很大帮助。

同时也要注意目录的命名，尽量精简，但不可随便填写，如“**aa**”,”**88**”。即使没有合适的命名，**Project** 也是最基本的，但必须在说明文档中说明程序内容。像学习过程中我们可以针对学习的主要内容而命名为”**1.Uart**”,”**2.LCD**”等等。如图 1、图 2 所示：

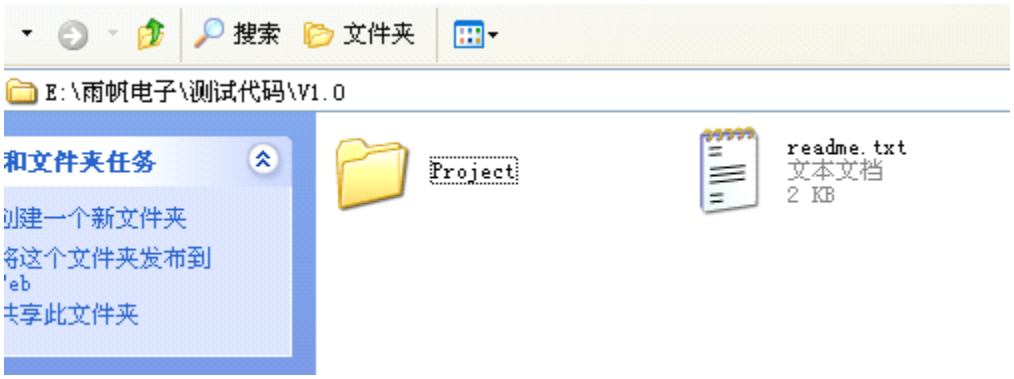

图 1、工程目录

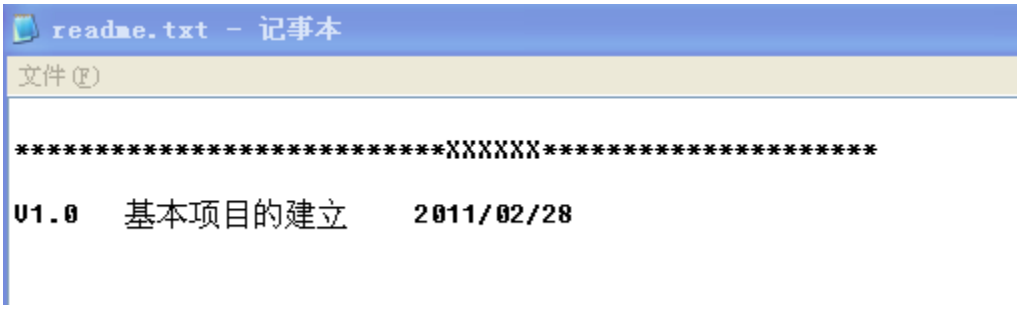

图 2、说明文档示例

然后开始生成项目。先在工程目录中建立如下 3 个文件夹：

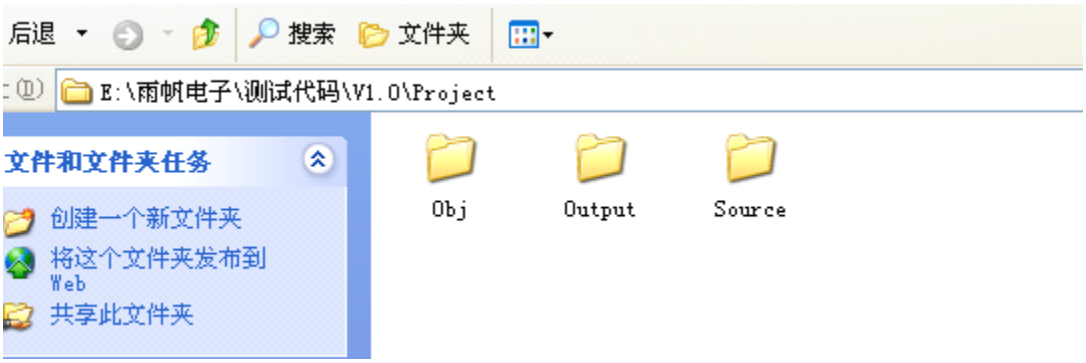

图 3、项目目录

打开 **Keil** 并填写项目名，这里我使用”**Project**”，保存在”**Obj**”目录中，并选择对应芯片。

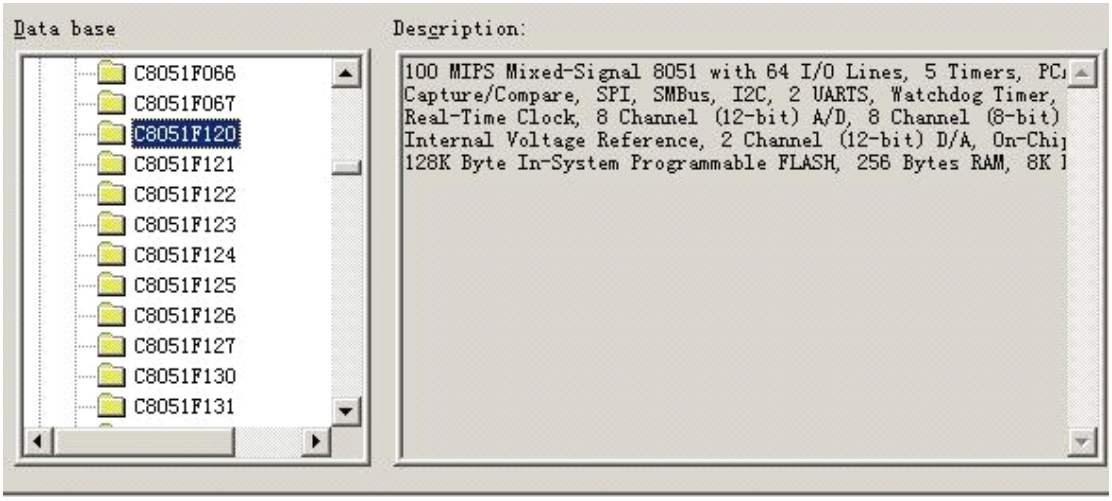

图 4、项目建立

完成后在弹出的是否添加 **Startup** 文件时选择“否“（C8051 单片机），然后点击 **Keil** 软件右上方的魔法棒状图标或点击”**Project->Options for Target ‘xxx’**”弹出项目设置对话框。

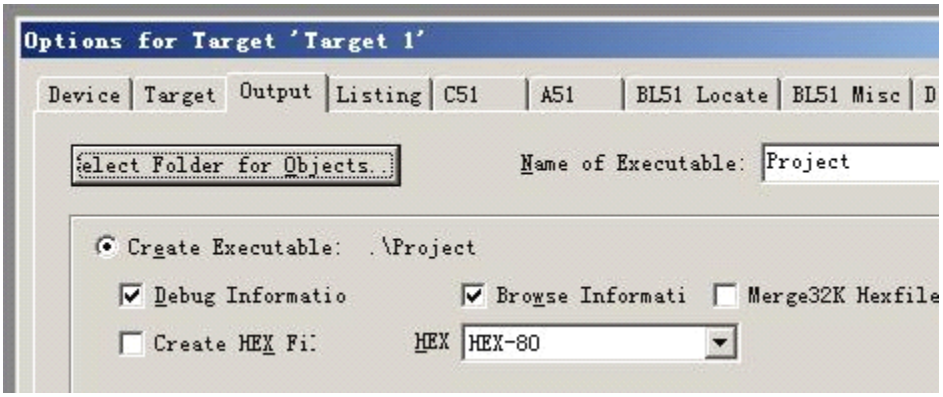

图 5、项目设置

将 **Output** 及 **Listing** 目录指向刚才建立的”**Output**”文件夹，这样就将编译时产生的文件放入指定的文件中，工程文件则在指定的”**Obj**”目录中，源文件放入 **Source** 目录中（这里我把 C 和 H 文件放入同一目录，也可分开，如 **inc, src**）这样整个工程看起来都清爽很多。

2.项目的结构

一个项目是由不同的模块文件所成的，有不少人只写一个 **main.c**….但对这样的程序来说，可移值性和可读性非常差。把不同的模块分开成单独的 C 文件和 H 文件是很有必要的，方便日后移值和维护。**main.c** 中建议只写系统的初始化代码及主函数 **main()**，其它功能函数分别写于不同的 C 文件中。

先看排版一：

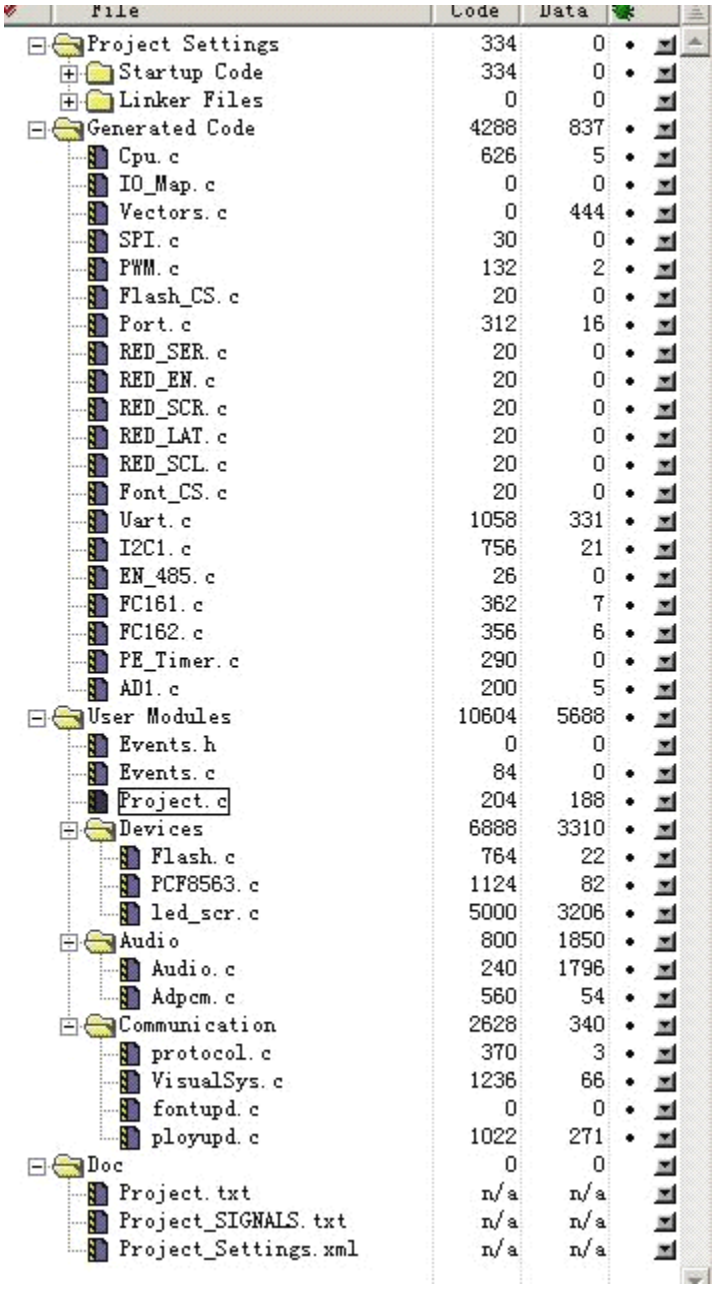

图 6、项目排版一

这是我用 **Freescale** 单片机写的一个程序，上面几个大的目录是编译器自动生成的，”**User Modules**”下的几个子目录则由自己建立。不同功能的子模块分开（对模块多而言，少的话不必），上面各个功能的模块分的很清，而且各个子模块文件的命名也很明了，一看就知各个模块功能，调试时很方便。而且，当我需要建立一个需要相同模块的工程时，我只需复制相应的 C 文件和 H 文件就可以使用了。

对于 **Keil51**，如下

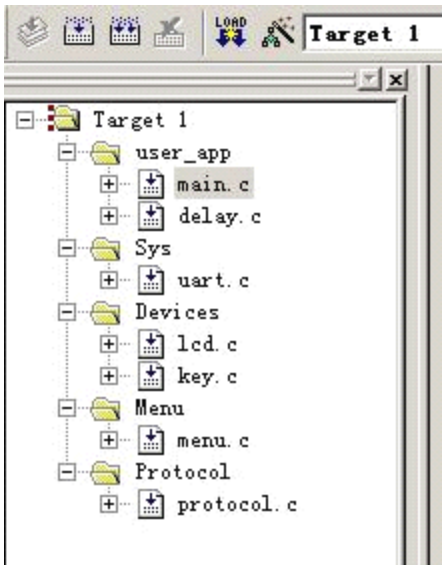

图 7、项目排版二

这里把各个功能都分开了，只是作为示范作用，所以一看略显分得过细。

尔后就有一个全局问题：当子模块函数较多时，各模块的子函数来回调用，就需要把所需要函数的头文件包含起来。这里我习惯两种方式：

第一种就是把头文件全部包含起来，当所含子模块较简单编译速度很快时很方便。建立一头文件”**includes.h**”，如下：

```
#ifndef     _INCLUDES_H
#define     _INCLUDES_H

#include    <C8051F120.H>
#include    "self_def.h"
#include    "protocol.h"
#include    "delay.h"
#include    "uart.h"
#include    "menu.h"
#include    "key.h"
#include    "lcd.h"

#endif
```

图 8、includes.h

其中 **#ifndef …**为条件编译，表示当没有定义 **_INCLUDES_H** 时则定义并包含以下头文件，这是为了防止重复定义。而包含头文件时”**< … >**”和” **”…“** ”的区别也得分清，一个是（优先）在编译器系统目录中查找，一个是（优先）在工程目录中查找。

这样我们在使用时，只需包含一个”**includes.h**”就可以调用全局的子函数（变量）了。

第二种是选择性包含，如果子模块中含有较复杂的成份而引起编译速度过慢时，只选择所需的头文件可减少编译时间。如下图：

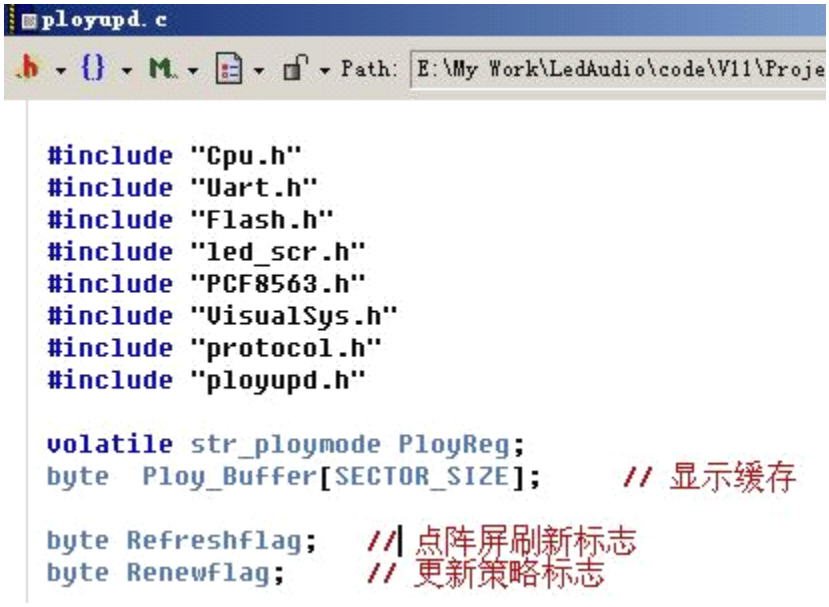

图 9、头文件选择性包含

而对于数据类型的定义，常用的有两种，一种是使用**#define** 宏定义。如：

```
#define   INT8U    unsigned char
#define   INT16U   unsigned int
```

另一种是使用 **typedef** 声明。如下：

```
typedef   unsigned char    INT8U;
typedef   unsigned int     INT16U;
```

这里我们要注意**#define** 和 **typedef** 的区别，最简单的就是顺序不同和有无”;”号。这是因为**#define** 是预处理指令，在编译预处理时进行简单的替换，简单说就是把被定义的指令复制过去，所以不能有”;”号，否则会将其“复制“过去。有时为了提高程序运行速度，把部分代码使用宏定义的方式，以牺牲代码空间来换取运行速度（代码段复制而非跳转）。但这样也有一个问题，因为只是简单的复制，所以当定义的函数中有变量传递时，最好加上括号，以防止运算符优先级等产生影响。

如定义一个乘法函数：

```
#define Multi(val)      val * 33
```

那我们调用时就是

```
tmpval = Multi(4) = 4 * 33;
```

而有时我们会加一个变量，像这样：

Tmpval = Multi(4+a) = 4+a*33

这样就知道哪里有问题了吧，应该这样写：

#define Multi(val)　　(val)*33

typedef 是在编译时处理的。它在自己的作用域内给一个已经存在的类型一个别名，如同定义变量的方法那样来声明一种类型。

推荐使用 **typdef** 来定义类型。而这里也因不同的习惯和编译器而用不同的定义，就对 **unsigned char** 而言，常见定义就有 **uchar, uint8, INT8U, UINT8, byte, u8…..**,这个还是看个人习惯为好。像我平时用 **INT8U**，而使用 **STM32** 时在 **MDK** 中就使用 **u8** 了，因为库中已定义好。

很多时间一个项目程序中对同种类型就有多个定义，这时为提高兼容性（真有时间也可以全改成一样的）可以专门定义一个头文件，我习惯命名为 **self_def.h** 或 **self_config.h**，然后包含于 **includes.h** 头文件中。

```
#ifndef _SELF_DEF_H
#define _SELF_DEF_H

#define     SELF_DEF
#define     DEBUG           1

typedef     bit             BOOL;
typedef     signed   char   INT8S;
typedef     unsigned char   INT8U;
typedef     signed   int    INT16S;
typedef     unsigned int    INT16U;
typedef     signed   long   INT32S;
typedef     unsigned long   INT32U;

typedef     signed   char   s8;
typedef     unsigned char   u8;
typedef     signed   int    s16;
typedef     unsigned int    u16;
typedef     signed   long   s32;
typedef     unsigned long   u32;

#endif
```

图 10、类型定义

3．函数及变量的命名规则

定义如下：

PPP 表示一外设的缩写，如：**UART**，**ADC**，**LCD** 等。外设函数的命名以外设缩写加下划线开头。每个单词的第一个字母都由英文字母大写书写。例如：**SPI_SendByte()**.这里我习惯在以单字符为单位时使用 **Byte** 或 **Char**，而使用双字节时使用 **Data**，而字符串则使用 **String** 或 **Str**。如：

UART_SendChar(INT8U casc)　　// 串口发送一字节
UART_SendString(INT8U *str)　　// 串口发送字符串

或 **Uart_....**

LCD_SendByte(INT8U dat)　　// 向 LCD 寄存器发送 8 位数据
LCD_SendData(INT16U val)　　// 向 LCD 寄存器发送 16 位数据

而 **PPP_Init()** 或 **PPP_Config()** 则表示名外设的初始化或配置。如 **LCD_Init()**、**SysClk_Config()**分别表示 **LCD** 初始化及系统时钟的配置函数。

每个函数都应有相应的注释，这个还是建议看相关的规范，因为我也有些取巧（C 文件和 H 文件也要相应的描述），只加了简单注释，当然是在不影响理解函数功能的前提下。像对比较复杂的函数，如各种算法及文件管理等，还是得详细写上注释为好。

对于变量名，也只是把常用的一些字母组合就可以了，即使 E 文不好也可以灵活使用。如数值，可使用 **dat, val..**等来表示，计数可以用 **cnt, cout, num** 来表示，**str, ptr** 表示指针，标志可用 **flag, bi**t 等来表示，而加上 **tmp** 或 **temp** 则作为中间变量做转换用，如 **tmpval, tmpdat…**

同时要区分开全局变量和局部变量。这里又有很多规范，不知道该听谁的…..我的做法是：

全局变量首字母大写，**Uart_InpBuffer[ ]**表示串口接收缓存， **Uart_OutBuffer[ ]** 表示发送缓存，也可以用 **Uart_RecvBuf[ ]**，**Uart_SendBuf[ ]**表示。**Uart_RcvLen** 则表示串口接收字符的个数（长度）。

局部变量用小写，而且简单写，不必做太多的组合。如 **k_dat,len ,val** 就可以分别表示按键值，长度数值，而 **i, j, k, cnt** 等则常用于计数，如 **for()**循环。

各变量也需加上简要注释，尤其是全局变量，方便编写调试程序及日后理解。

4．端口定义

对于端口定义，也需要合适的命名以方便理解。如：

```
//********** 端口定义 ***********//
sbit    LCD_CS          = P3^0;         //  LCD片选端口
sbit    LCD_RES         = P3^1;         //  LCD复位
sbit    LCD_DC          = P3^2;         //   0   指令   /   1   数据
sbit    LCD_SCL         = P3^6;         //  时钟
sbit    LCD_SDA         = P3^7;         //  数据
```

图 11、端口定义

而对于 C8051 中有的 IO 口不能采用位操作的方式，则使用以下规范：

端口置高时使用端口名加上 **SetVal()**，清零时则加上 **ClrVal()**，读取端口值时则使用 **GetVal()**，如下：

```
//********** 端口定义 ***********//
#define LCD_CS_SetVal()    (P6 |=  1 << 0)    // CS ~ P6^0
#define LCD_CS_ClrVal()    (P6 &=~(1 << 0)
#define LCD_RES_SetVal()   (P6 |=  1 << 1)    // RES~ P6^1
#define LCD_RES_ClrVal()   (P6 &=~(1 << 1)
#define LCD_DC_SetVal()    (P6 |=  1 << 2)    // DC ~ P6^2
#define LCD_DC_ClrVal()    (P6 &=~(1 << 2)
```

图 12、端口宏定义

5．代码书写规范

1. 代码对齐，包括首字对齐和括号对齐，缩进固定为 4 个空格(**Table** 键).。
2. 运算符前后添加一个空格，当为整个版面整齐可以运算符之间以空格补齐。如：

```
INT16U tmpval = 0;
INT8U  len    = 0;
```

3. **if / for / while / do** 等各占一行，执行语句不得跟随其后。

```
for(i = 0; i < 200; i++)
{
    len++;
    val++;
}

cnt = 0;
do
{
    ....
    cnt++;
}while(cnt < 200);
```

4. 一行代码只实现一个功能。

5. **switch** 函数一定要有 **default 分支**，以防止程序运行过程中出现意外。

6. 对运算符之间使用必要的括号，以防止优先级出错。

7. 函数的参数和返回值没有时使用 **void**，如：

```
void ADC_Init(void)
{
    ----
}
```

8. 常数及表格使用 **code** 或 **const** 修饰。

9. 各函数之前有且只有一空行，各模块函数起始和结尾同样为一空行。

至此，我们已建立好一个符合“标准”及个人习惯，同时方便日后移值维护的工程。开发板中的各个例程也将按照以上规范书写，同时配有相关的文档，对例程中的要点难点及注意事项进行详细说明，以方便理解和学习。

此规范由网友 **amazing030** 提供在此表示感谢！

雨帆电子

2011.2.28