

# KF8CC 用户使用注意手册

V1.3

---

目录

|                             |    |
|-----------------------------|----|
| 1 概述.....                   | 3  |
| 2 关键字与保留字.....              | 3  |
| 3 BIT 类型.....               | 4  |
| 4 数据类型.....                 | 5  |
| 5 字符数组限制.....               | 6  |
| 6 全局指针限制.....               | 6  |
| 7 数组访问.....                 | 7  |
| 8 全局变量和地址指定.....            | 7  |
| 9 VOLATILE 关键字.....         | 7  |
| 10 STATIC 关键字[调试模式不支持]..... | 8  |
| 11 递归函数[不支持].....           | 8  |
| 12 函数不可重入.....              | 8  |
| 13 不支持内联函数.....             | 10 |
| 14 不支持函数指针.....             | 10 |
| 15 函数参数及限制.....             | 10 |
| 16 汇编嵌套.....                | 10 |
| 17 中断函数.....                | 11 |
| 18 类型转换.....                | 12 |
| 19 头文件限制 [变量定义限制].....      | 12 |
| 20 调试编译.....                | 12 |

## 1 概述

kf8cc 编译器为 KF8 系列的单片机提供了一个 C 语言程序开发平台。本文档针对 kf8cc 使用注意事项做简单说明，以便用户可以很快掌握如何使用 kf8cc 开发程序。

## 2 关键字与保留字

标准 C 保留字

表 2-1

|        |        |        |          |         |          |          |
|--------|--------|--------|----------|---------|----------|----------|
| auto   | double | int    | struct   | break   | else     | long     |
| switch | case   | enum   | register | typedef | char     | extern   |
| return | union  | const  | float    | short   | unsigned | continue |
| for    | signed | void   | default  | goto    | sizeof   | volatile |
| do     | if     | static | while    |         |          |          |

kf8cc 保留字

表 2-2

|           |            |             |            |        |
|-----------|------------|-------------|------------|--------|
| at        | _at        | __at        | code       | _code  |
| __code    | critical   | __critical  | __critical | idata  |
| _idata    | __idata    | data        | _data      | __data |
| interrupt | _interrupt | __interrupt | sfr        | _sfr   |
| __sfr     | _using     | __using     | __using    |        |

kf8asm 保留字

表 2-3

|      |     |  |  |  |
|------|-----|--|--|--|
| high | low |  |  |  |
|------|-----|--|--|--|

kf8cc 保留字用作功能扩展使用，用户定义的标识符不能与其重名。

C 开发在建立结构体、联合体时不支持符号 **high** 和 **low** 定位数据的高地位（**调试不支持**）。如

```
typedef struct{
    unsigned char high;
    unsigned int low;
```

```
}test; 此定义 release 编译通过，但 debug 编译失败，即语法  .def high, 识别为内部函数，high 在汇编代码中正常示例如：MOV R0,#high( a) MOV R0,low( a)
```

### 3 Bit 类型

kf8cc 当前版本未实现 Bit 类型,但 kf8cc 可以使用标准 C 的位域来实现 Bit 操作。

例如:

```
typedef struct
{
    unsigned char b0:1;
    unsigned char b1:1;
    unsigned char b2:1;
    unsigned char b3:1;
    unsigned char b4:1;
    unsigned char b5:1;
    unsigned char b6:1;
    unsigned char b7:1;
}BitStruct;
BitStruct FlagColumn;
#define      flag0      FlagColumn.b0
#define      flag1      FlagColumn.b1
#define      flag2      FlagColumn.b2
#define      flag3      FlagColumn.b3
#define      flag4      FlagColumn.b4
#define      flag5      FlagColumn.b5
#define      flag6      FlagColumn.b6
#define      flag7      FlagColumn.b7
```

这样就定义了 8 个 Bit 位。也可以使用头文件系统中类型定义。

```
typedef struct
{
    unsigned char _0:1;
    unsigned char _1:1;
    unsigned char _2:1;
    unsigned char _3:1;
    unsigned char _4:1;
    unsigned char _5:1;
    unsigned char _6:1;
    unsigned char _7:1;
}sfr_bits;
```

建议中断需要未状态标记与主程序中状态标记分开,即定义在不同的字节对象中。

## 4 数据类型

| 数据类型           | 位长 |
|----------------|----|
| char           | 8  |
| unsigned char  | 8  |
| short          | 16 |
| unsigned short | 16 |
| int            | 16 |
| unsigned int   | 16 |
| long           | 32 |
| unsigned long  | 32 |
| float          | 32 |

立即数数据显示类型表达:

```
// [unsigned long]T2_cnt =PWM5L1<<8u; // 正确
// [unsigned long]T2_cnt +=PWM5L0;
```

```
// [unsigned long]T2_cnt =PWM5L1<<8; // 错误
// [unsigned long]T2_cnt +=PWM5L0;
```

即立即数支持类型修饰: u 为无符号, f 为浮点, l 为 long 型。ul 为 unsigned long 型。

上述错误分析: PWM5L0 unsigned char 操作 signed int 8, 即将立即数识别为有符号的, 故 PWM5L0<<8 的结果类型提升为 signed int, 当转换到结果的 unsigned long 时, 根据结果的高位, 即 bit15 识别数据符号, 根据符号将 unsigned long 的高 2 字节赋值为 0 或 0xFF, 即当 PWM5L1 大于等于 128 时, T2\_cnt 值错误, 增大 0xFFFF0000。

kf8cc 支持结构体和联合体, 仅结构体支持赋初始值。参考示例如下:

```
typedef struct // 自定义协议结构体
{
    uint8 id;
    uint8 type;
    uint8 product[6];
    uint8 index[5];
    uint8 reservation[DATAFLASH_BUFFER_BYTE_SIZE-13]; // 前面占
    用字节数参与, 如这里的13
}s_Dataflash_ConstBuffer;
const s_Dataflash_ConstBuffer __at(0x1800) appUserData={
    .id=8,
    .type=1,
    .product={1,2,4,5,6,7,8,9,10},
    .index="abce\0",
    //.index={'a','b','c','e'},
}
```

```
.reservation={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,}  
};          // 需要注意的是 常量在内容中1个flash空间字只能存放1个字节，所有不能  
直接到RAM变量空间等效传递。
```

如果需要定义联合体的多种结构，可以分别定义内容结构体，即内容结构体对象意义下赋初值。联合体对象或参数负责差异代码编写。如下联合体定义：

```
typedef union  
{  
    unsigned char bytes[DATAFLASH_BUFFER_BYTE_SIZE];    // 无协议结构下的  
    纯数组大小缓存  
    struct // 自定义协议结构体 服务  const对象  
    {  
        uint8 id;  
        uint8 type;  
        uint8 product[6];  
        uint8 index[5];  
        uint8 reservation[DATAFLASH_BUFFER_BYTE_SIZE-13]; // 前  
        面占用字节数参与，如这里的13  
    };  
} s_Dataeeprom_RAMBuffer;          // 与 const绑定的 读写样例支持，数量类型提示，  
仍按字节赋值，高位留空即实现原始读取Flash的高位CRET的0xB0占位
```

## 5 字符数组限制

kf8cc 已支持字符串数组。

kf8cc 当前版本对数组初始化使用以下格式：

```
char ch[]={ '0' , ' 1' , ' 2' , ' 3'  };  
或 char ch[]={ "0123" };  
需要注意的是字符串末尾会放置 0，但字符串不支持\0 的间隔多表达，如  
char ch[]={ "01\023" };此时失败字符串仅长度 3 的初始 0x30 0x31 0x00
```

当字符串位于结构体数据类型时，初始应该严格满足其定义的长度，即不能多也不能少，否则整体结构体数据初始化存在错误偏移。

## 6 全局指针限制

kf8cc 支持指针，但不支持全局指针赋初值。

`unsigned char *p=&var;` 错误

编译结果丢弃初值，即 p 的地址为 0x000000 的异常，正确写法

全局定义 `unsigned char *p;` 在启动初始化函数中 `p=&var;`

## 7 数组访问

考虑到 kf8 系列单片机的 bank 特性，任何非 const 类型数组占用空间不得超过一个 bank，例如 char 类型数组，其下标值不得超过 128，int 类型数据，其下标值不得超过 64。

在使用数组时尽可能使用 unsigned char 类型数组和 unsigned char 类型下标。

通过表达式计算数组下标时，用户需要自己确认表达式的值是否超过该表达式类型的限定值，否则将会造成溢出的错误。

Kf8cc 支持一维和二维的数值，当相比较而言，二维的效率偏低。建议使用一维数组，针对二维需要，可以采用显示表达式方法实现，如 `AA[i][j]=0x05`，可以写为 `AA[i*Maxj+j]`，针对数组的连续代码操作，可以将下标定位作为独立变量写在前面。如 `u8 b= i*Maxj+j; AA[b]=AA[b+1];` 比 `AA[i*Maxj+j]=AA[i*Maxj+j+1]` 效率较高。

## 8 全局变量和地址指定

kf8cc 支持常量和非常量的全局变量，并支持声明时进行初始化。

常量变量只能全局声明，即不能在函数内部建立常量变量。否则会造成错误的结果。

变量的声明在 C 语言中会自动分配 RAM 空间，如果需要，可以指定地址，指定方法为在类型后面，名称前面添加。如 `unsigned char __at(0x83) a=4;` 即建立了 C 变量，并 a 的 RAM 地址为 0x083。如果需要 a 指向 RAM 分区的 3 区时，可以写为 `unsigned char __at(0x383) a=4;` 如果为常数变量，可以写为 `const unsigned char __at(0xF70) a=4;`

变量的建立针对 kf8cc 只能建立在 c 类型的文件中，如果需要其他文件使用，可在其对应的 h 文件或总的 h 文件中进行声明，即 `extern unsigned char __at(0x83) a;`

## 9 volatile 关键字

一般情况下不需要使用 volatile 修饰，如果变量同时在中断和主循环中

使用，需要添加该关键字修饰，即 2 条独立分支的改变关系不确定，涉及该变量的代码表达不应使用优化后的识别结果。

## 10 static 关键字[调试模式不支持]

kf8cc 支持 static 关键字。

static 有以下用法：

- 1) 修饰函数，表示函数在定义它的 C 文件外不可访问
- 2) 修饰全局变量，表示在定义它的 C 文件外不可访问
- 3) 修饰局部变量，表示此变量位于静态存储器。此变量在函数退出后依然存在
- 4) 对于 static 修饰局部变量时，可以使用全局变量代替。不建议使用 static 修饰函数，该限定针对调试编译存在信息异常情况，编译将会识别。如果仅限文件使用函数不对外声明即可

## 11 递归函数[不支持]

kf8cc 不支持递归函数，因局部变量唯一。KF8 系列芯片 PC 指针通过硬件堆栈保存程序嵌套不可以超过硬件堆栈大小，硬件堆栈大小分 8 级和 16 级两种。

故函数不能同时在高和低中断代码中调用。也不能在高级中断或低级中断，并和主程序中同时调用。默认冲突 kf8cc 会报错并给予文字提示，当链接器使用 -z 选项下，将可能的函数冲突做警告提示。

## 12 函数不可重入

kf8cc 编译器产生的代码不可重入。

由于函数不可重入，以下操作将产生问题：

- 1、中断中调用函数外部函数
- 2、中断中进行\*、\、%以及指针操作
- 3、中断函数的参数传递

如果要在中断中调用函数，不建议这样做。参见函数使用限制章节。

- 1、调用无参数的函数（可以通过全局变量传参）
- 2、手动保存传参堆栈

注意：此函数不能在中断外调用，在外部调用时需要进入临界段（关闭中断）



例子:

1、调用带参数函数(中断外部不调用)

```
void ISR() __interrupt
{
    if(T0IF)
    {
        PUSH_STK();//保存传参堆栈

        //中断内容
        //可以调用带参函数，但是函数不能在中断外调用
        fun(arg0,arg1,.....);

        POP_STK();//恢复传参堆栈
    }
}
```

保护堆栈函数应该自行定义，数量已函数参数数量为准，参数传递依次使用 R0, STK00, STK01...

2、调用带参数函数(中断外部调用)

```
void ISR() __interrupt
{
    if(T0IF)
    {
        PUSH_STK();//保存传参堆栈

        //中断内容
        //可以调用带参函数，但是中断外调用时需要关中断
        fun(arg0,arg1,.....);

        POP_STK();//恢复传参堆栈
    }
}
```

.....

```
SAIE=AIE;           //保存中断总控制位
AIE=0;              //关中断
fun(arg0,arg1,.....);
AIE=SAIE;           //恢复中断总控制位
.....
```

该方法同样适合无参数函数，建议针对外部和中断内部同时使用的函数，仅函数书写 2 遍，命名不同的内容，即独立使用。

## 13 不支持内联函数

## 14 不支持函数指针

## 15 函数参数及限制

kf8cc 函数参数长度有限制，函数所有参数的长度不可以超过 13 个字节。参数根据字节使用数量，依次使用 R0、STK00、STK01、STK02、STK03...STK11。中断压栈不保护 STKxx 变量。注意事项见函数不可重入章节。

当只有一个 char 或 uchar 类型的参数时，传参仅使用 R0，此时不需要考虑中断调用问题。如果为 int 时使用 R0 和 STK00，其中 R0 存放参数的高位。如果为 long 型或者 2 个 int 型时，按照顺序使用 R0、STK00、STK01、STK02，需要注意的是低字节使用大编码的变量，如 int 型使用 STK04、STK05，STK04 里面存着变量的高字节内容。

函数返回同样使用该参数，规则相同，即结果可以按照参数考虑。根据需要占用的字节数，将内容缓存到对应的变量中。

Kf8cc 不支持函数在中断和外部同时使用下的自动建立备份，即该用法会产生错误（过程变量作用时，被中断打断，中断运行重新使用了过程变量，中断结束外部代码的过程变量已被改变，后续代码执行会错误）。

不建议使用 static 修饰函数，调试编译不支持。

注意：乘法、除法、浮点、指针均是按照库的形式实现，本质为函数，因此不能在中断里面、包括中断调用的函数存着这些运算。除法外部主程序未使用，但需要考虑外部函数调用参数传递的堆栈变量保护，见函数参数章节说明。

## 16 汇编嵌套

kf8cc 汇编嵌套语法为：

```
__asm  
汇编代码  
__endasm;
```

在汇编代码中要访问寄存器和全局变量时需在变量名前加下划线“\_”，嵌汇编不支持局部变量的使用。

例：

```
__asm
Label:
    MOV R0, _P0
    MOV _a, R0
    MOV R0, _P1
    MOV _P0, R0
    MOV R0, _a
    MOV _P1, R0
    JMP $-6    // 或 JMP Label
__endasm;
```

注意：

- 1、嵌汇编可以使用标号，**标号后面需要添加“:”**
- 2、建议嵌汇编使用 R 寄存器仅使用 R0、R1，如果需要可以使用 R6 和 R7。R2~R5 被编译器用来做中断保护现场的压栈出栈使用。除非关闭了中断，否则 **R2~R5 不能作用在嵌汇编代码中**。
- 3、针对定义的全局变量的嵌汇编表达，变量存在分区信息。而编译器会根据需要进行切区的优化，但必须是基于伪指令进行优化，使用举例如下：

```
(unsigned int) var++;
```

```
__asm
    BANKSEL _var
    INC _var
__endasm;
```

- 4、针对存在 MOVP 指令的型号，嵌汇编调用函数也需要进行伪指令说明，有编译器进行切区代码的实际实现，使用举例如下：

```
Fun(a, b);
__asm
    ; 传递参数 a,b 到参数堆栈，使用见函数参数章节。
    PAGESEL _Fun
    CALL _Fun
    PAGESEL $
__endasm;
```

- 5、如果函数带有返回内容，**返回结果所在**寄存器类型参数的使用，见函数参数章节的说明。

## 17 中断函数

KF8CC 中断函数格式如下：

```
void int_fun0() __interrupt (0)
```

```
void int_fun1() __interrupt (1)
```

函数名可以换做任何用户喜欢的合法的标识符。

其中后面括号内的 0 和 1 分别代表高级中断入口 0x04 和低级中断入口 0x14。

需要注意的是 C 项目代码不需要自己书写压栈指令，编译器会自动处理。压栈会保存内容到 R2~R5 和建立的全局变量中。也就是说中断函数代码除了书写的内容外还存在压栈和出栈的额外代码。如果调用函数，使用的参数需要手动代码压栈出栈处理。

## 18 类型转换

例如：

```
unsigned char Result = 0;
unsigned char FristNum ;
unsigned cahr SecondNum;
Result = FristNum * SecondNum;
```

如果两个数相乘结果大于 255 需要将乘数与被乘数类型转换成 int 型或者 long 型，即需要类型转换，修饰采用括号内类型的表达形式，例如：

```
unsigned int Result = 0;
unsigned char FristNum ;
unsigned cahr SecondNum;
Result = (unsigned int)FristNum * (unsigned int)SecondNum;
```

## 19 头文件限制 [变量定义限制]

头文件只能有变量或函数声明。即变量只能声明在源文件中，如果变量需要外部使用，只需要在头文件中添加 extern 修饰进行引用，其他文件包含该头文件即可。

## 20 调试编译

KF8 系列单片机调试使用软件调试，即调试编译下需要添加额外的监控代码，同样需要占用一定的 RAM 资源，一般情况下调试代码占用小于 255 句，RAM 占用少于 16 字节。

针对 C 项目，不需要考虑 RAM 使用情况，即调试变量等同全局变量，地址分配不存在冲突，指定变量地址情况除外。

使用汇编项目时可以采用申请变量的伪指令编写方法，更多的汇编开发变量为定义 RAM 地址，使用调试编译时应该考虑调试代码占用 RAM 情况，代码变量不应与这些地址重合。因调试变量存在，调试编译会提示 “warning: call \_cinit at startup to clean bss section.”，除非使用了申请变量并赋初值的伪指令



## KF8CC 用户使用注意手册

声明，否则不需要考虑调用。针对不同的系列的单片机使用的调试 RAM 资源存在差异。具体可以通过编译后的后缀 map 文件查看，具体在 Symbols 段中，变量以“ICD”作为前缀，使用在 xxx\_monitor.asm 文件中。