

实现单片机复位不清零 RAM 的方法

在产品的实际运行环境中，可能存在系统电源的异常波动，造成电压低于芯片的可靠工作电压，从而芯片复位重新运行来确保指令的正常工作。针对部分特殊产品应用，运行过程中需要状态保持或整定的参数保留时，可以在芯片工作前判断是否欠压复位，当有该需求时，可以采用有差别的进行初始化处理。这也是本篇文章介绍所有使用的方法。

为了实现这一目的，首先我们要了解编译器的工作原理，当用户定义带初值的全局变量时，编译器会隐性的在代码运行开始将对应的 RAM 地址赋值。因此一旦代码生成每次上电该变量均为被赋初始值，同时这段隐形代码还负责了其他 RAM 的初始值或清零。要想达到复位不处理 RAM 的目的。可以基于 2 中思路：1)、编译器提供参数接口，不形成初始值和清零命令。2)、利用现有的实现模式，重构初始代码的实现。

如果采用第一种思路，编译器提供参数，则初始化过程 RAM 不清零、不赋值。当如定义全局变量时代码为 `unsigned char Val_Value=255;` 实现模式即为在初始化 RAM 函数中将对应地址写入数据 `0xFF`，而参数的选择后编译器处理该变量不赋值，实际上首次上电还是需要该初始值来指导程序工作，而该需求下的上电复位和低电压复位编译器无法区分，因此还需要用户的初始化代码部分做跳转代码根据情况选择。在这种模式下，用户需要去修改编译器的入口给出参数，该项操作自身比较麻烦，还要在代码实现过程中还考虑是上电运行还是复位运行，要求代码全局变量也不能写为 `unsigned char Val_Value=255;` 而只能是 `unsigned char Val_Value`，在上电运行逻辑的初始化过程加入显式代码赋值 `Val_Value=255`。对比可知第二种思路下，仍旧考虑初值代码的实现，而不需要更改编译器的参数，毕竟修改后二次开发还要参数改回。额外要做的事情只是在代码中加一个固定空函数，二次开发中不需要这种应用的话直接屏蔽即可。

综合比对后，第二种更能满足产品设计的需求。关于重构的实现，针对原理做以下说明（仅针对 C 语言，汇编需要完全的代码自行编写控制）。对一个 C 语言程序来说，他的入口就是 `main` 函数，而 `main` 函数一定不是存放在程序运行的首地址（约定）。单片机程序的执行一定从首地址开始，因此需要加入跳转指令到 `main` 函数所在的地址。除了处理代码的编译外，数据的初始化也是编译器要处理的重要部分，而且要在程序运行之前，不然初始的需求还没实现，代码已经使用就达不到所需的效果。需要在跳转到 `main` 函数之前一定执行一段代码实现 RAM 的初始化处理，而该处理不需要用户考虑，有工具自动加入。粘贴如下代码说明：

```
STARTUP.code 0x0000
NOP
PAGESELinit_imp
JMP    init_imp

.global init_imp
```

```
INIT_IMP      .code
init_imp
PAGESEL_startup
CALL          _startup
PAGESEL__gsinit_startup
CALL          __gsinit_startup
PAGESEL_main
JMP           _main
```

由以上代码可以看到编译器在开始从 main 函数运行之前执行了 2 个子函数。而这 2 个子函数的目的就包括 RAM 的初始化处理，另一就是加载校准信息。

芯片的数据手册指出运行只有加载了校准信息时才能保证内部的时钟或 LDO 是准确的，编译汇编程序时这段代码必须需要我们自己编写或复制粘贴，C 语言编译器自动添加了调用代码，能够忽略芯片差异，使我们重点着手系统代码的开发。虽然如此，我们还是可以编写自己的调用校准代码，即在 C 源文件中建立一个函数 `void startup () {}`。在函数内可以编写自己的校准信息加载方法，可以嵌汇编也可以完全的 C 语言。同样原理我们可以建立一个函数，即

```
void _gsinit_startup ()
{
}实现 RAM 的手动代码编写。
```

这里做原理说明如下：

一个函数前面加了一个下划线，一个不加，是因为编译器对代码段名字的约定，见上面代码示例说明。C 语言的函数编译后会在前面加一个下划线。因此写的函数编译后满足逻辑所需，可以用来替换库或链接器自动生成的方法。而这个替换原则是优先顺序法则。一个函数就是一个段，编写的源码段就是第一优先级，而库或编译器生成的函数在第二优先级。当源码中涉及后，编译调用的实体也就变成我们重新写的代码。

过程原理明白后我们就可以着手编写直接的初始化代码，结合上电标志即可满足复位运行不清零 RAM 的需求，参考示例代码如下。

```
unsigned char i;
//unsigned char i=200; 赋值失败
// 空函数，使编译器结果对RAM不作为
void _gsinit_startup()
{
}
// 手动上电复位下RAM初始值处理
void user_init_ram()
{
    i=200;
}
//主函数
void main()
{
```

```
if(POR&&LVR==0) // 非上电复位下的 欠压复位
{
    LVR=1; // 可继续监听欠压复位
}
if(POR==0) // 上电复位标志
{
    POR=1;
    SLVREN=1; // 默认状态为开
    LVR=1; // 可接受欠压复位标志
    user_init_ram();
}
//=====
init_mcu(); // 功能初始化
while(1)
{
    // 添加自己的代码
}
}
```

该文指出了实现该功能的工具方法介绍，但该需求的实现应注意一些事项，只有满足时才能采用这种方法。

- 1) 全局变量定义时不能赋初始值;
- 2) 芯片功能不应在一个状态下切换冲突（复位时寄存器值更多的复位或不确定，基本要重新设定）;
- 3) 上电标志的逻辑要处理正确（正确识别断电上电和欠压上电）;
- 4) 配置字需配置需开启压检测功能，即LVREN=1。建议打开上电延时功能。
- 5) 必须放置空函数 **gsinit_startup** 使RAM初始化工作不作为。
- 6) 欠压不能超多一定的量，否则电气特性上会识别成上电运行，不能保证RAM的值状态，需要重新初始化。