**武汉大学国家网络安全学院**

**本科生实验报告**

**《嵌入式系统实践》**

学 院 名 称 ：国家网络安全学院

专 业 名 称 ：信息安全

课 程 名 称 ：嵌入式系统实践

指 导 教 师 ：丁玉龙

姓 名 ：夏云琨

学 号 ：2021302141191

2024.06.25

目录

[1 功能描述 3](#_Toc170653360)

[1.1 基础功能 3](#_Toc170653361)

[1.2 安全设计 3](#_Toc170653362)

[2 功能设计 4](#_Toc170653363)

[2.1 main函数 4](#_Toc170653364)

[2.2 Clear\_Display函数 6](#_Toc170653365)

[2.3 Receive\_Number函数 6](#_Toc170653366)

[2.4 Receive\_Calculating函数 7](#_Toc170653367)

[2.5 Result\_Handle函数 7](#_Toc170653368)

[3 安全设计 9](#_Toc170653369)

[3.1 数据单元备份与校验 9](#_Toc170653370)

[3.2 输入滤波 11](#_Toc170653371)

[3.3 输出刷新与配置刷新 11](#_Toc170653372)

[3.4 看门狗 12](#_Toc170653373)

[3.5 前序代码检查 13](#_Toc170653374)

[3.6 上电延时与冷热启动 15](#_Toc170653375)

[3.7 睡眠躲避干扰 15](#_Toc170653376)

[3.8 随机延时 16](#_Toc170653377)

[3.9 随机乱序执行 17](#_Toc170653378)

[3.10 超时判错 18](#_Toc170653379)

[4 功能展示 19](#_Toc170653380)

[5 实验感想 22](#_Toc170653381)

# 功能描述

## 基础功能

实现一个简单的计算器，能够进行简单的加减乘除运算，实现方式如下：

·数字0~9分别表示输入0~9，ABCD分别表示加减乘除，\*表示=，#表示清空当前输入。

·计算器能够接受两个参数，当输入长度或运算结果长度超出显示限制即8位十进制时，会忽略。

·用户操作步骤如下：

1. 按数字键输入第一个参数
2. 按ABCD任意一个选择运算模式
3. 输入第二个参数
4. 若出现输入错误，按#清空
5. 按\*进行运算
6. 数码管显示运算结果
7. 按任意键清空，进行下一轮运算

## 安全设计

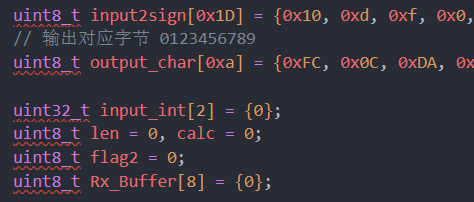
1. 数据单元备份与校验
2. 输入滤波
3. 输出刷新与配置刷新
4. 看门狗
5. 前序代码检查
6. 上电延时和冷热启动
7. 睡眠躲避干扰
8. 随机延时
9. 随机乱序执行
10. 超时判错

# 功能设计

功能代码主要流程为：

* 1. 读取用户输入
  2. 将用户输入转为操作码
  3. 根据操作码进入对应处理函数

我们设置了以下全局变量，其中input2sign用于将用户输入转为操作码，转换后0~D对应操作码0x0~0xD，\*对应0xE，#对应0xF，便于后续处理。output\_char用于将要显示的数字转换为对应的数码管字节；input\_int用于存储两个参数；len用于保存目前正在接受的参数的十进制长度；calc用于保存运算符；flag2用于标志目前程序运行的阶段；Rx\_Buffer用于记录目前数码管显示内容。



下面逐函数进行功能代码的介绍。

## main函数

main函数主要实现初始化和读取键盘内容。

1. 启动初始化

主要包括硬件初始化、冷热启动、备份校验部分。





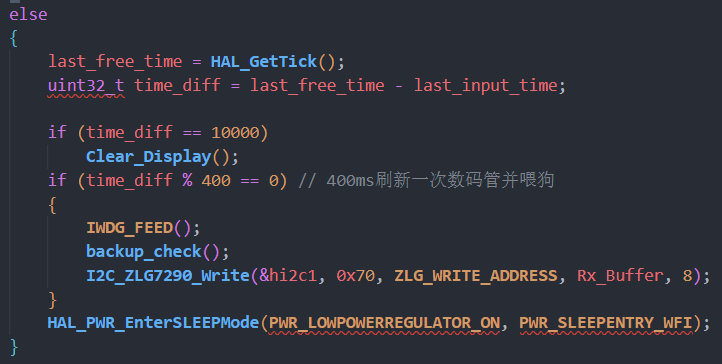
1. 循环接收用户键盘中断

在while循环中，判断当前是否产生键盘中断。当中断到来，全局变量flag1的值将会被置为1，可根据flag1进行判断。

（1）键盘输入处理：读取键盘输入后，若键盘输入不合法，便会忽略此次输入；若合法。则将其转为操作码。如果此次输入为#或者flag2大于等于2即已经进行了一轮运算，便清屏。然后再根据操作码调用对应的处理函数：

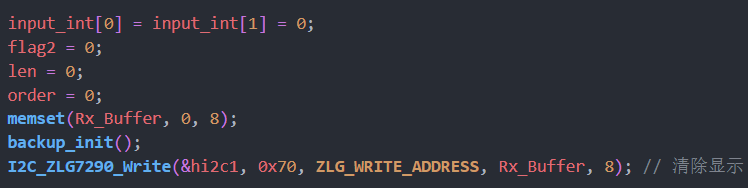


（2）else分支：该分支表示当前无键盘输入，进行周期性备份校验和数码管显示刷新以及睡眠。



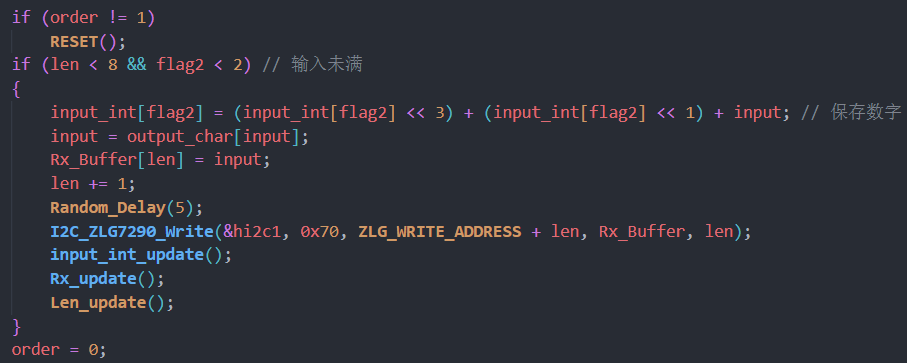
## Clear\_Display函数

Clear\_Display函数主要功能为将计算器还原到初始状态，并清除显示：



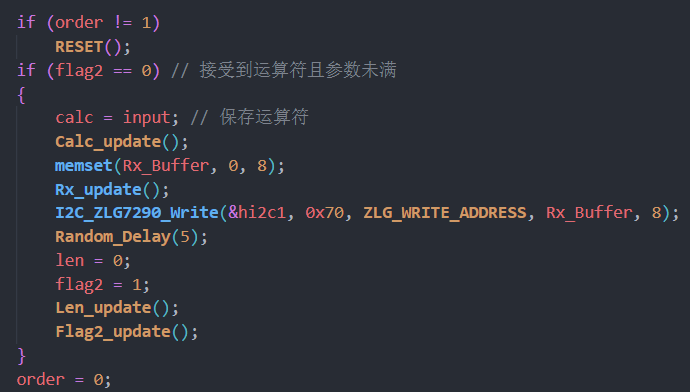
## Receive\_Number函数

该函数功能为处理输入的数字。当flag2为0时表示目前正在接受第一个参数，为1时表示正在接受第二个参数。在当前参数长度小于8且当前参数量不超过2个时进行处理：对当前参数乘十并加上输入的数字，当前参数长度加一，并显示当前参数。



## Receive\_Calculating函数

该函数功能为处理输入的运算符，具体实现如下：当flag2为0即当前已接收过第一个参数时进行处理。保存运算符，清空数码管显示，当前参数长度置0，将flag2置1即当前正在接受第二个参数。



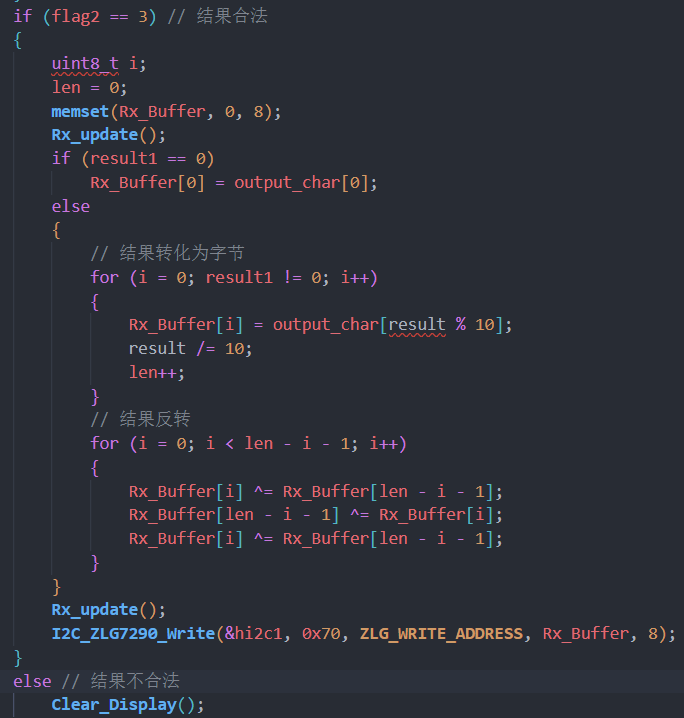
## Result\_Handle函数

此函数功能为计算结果和结果处理，在flag2为1即接收第二个参数时可以运行，具体实现如下：

计算部分：将flag2置2，表示结束参数读取，进入计算。根据保存的运算符进行计算，结果合法时，flag2加一。由于数码管显示原因，将结果设为不能超过8位十进制且不能为负。加法合法条件为结果小于最大值；减法为参数1大于等于参数2；乘法为两数相乘小于最大值，除法为除数不为0。



结果处理部分：将结果逐位转成输出字符，由于输出时低索引处为高位，故还需倒转Rx\_Buffer数组，然后将结果输出。

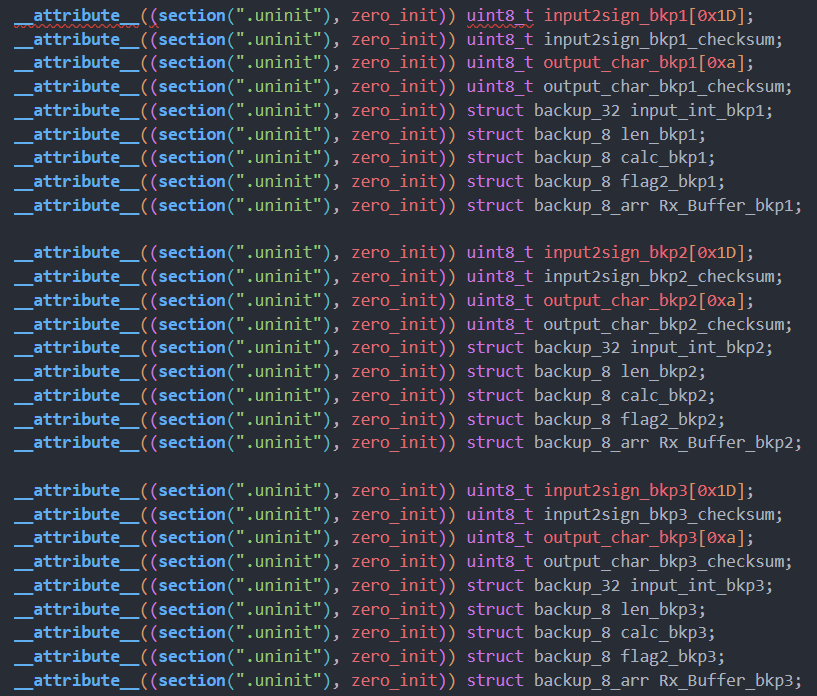


# 安全设计

## 数据单元备份与校验

在STM32中存在IRAM1与IRAM2，将IRAM1设为默认，用来存储代码，IRAM2存放重要数据，并将设置为不初始化，这样每次reset都不会初始化IRAM2中的数据。将这片不可初始化的区域命名为.uninit。

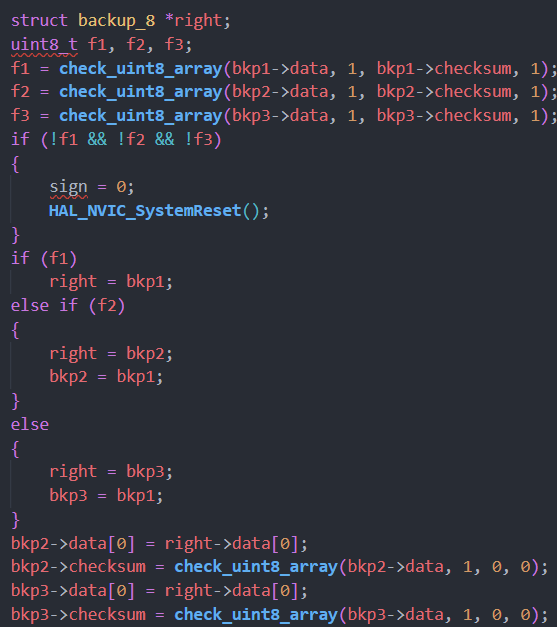
为先前提到的全局变量创建三个备份，放入.uninit区域，将相同数据的不同备份分开存放：



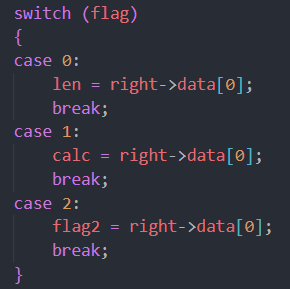
·备份校验

由于备份类型较多，校验函数也较多，这里选择其中一个来解释流程。

首先对三个备份进行校验，数组的校验为互相异或之后再异或一个特定值，单个数据校验为直接异或特定值。然后对三个校验结果进行判断，若三个备份都不对，进入冷启动。只要有一个校验结果正确，就将该备份复制到另外两个备份并重新计算校验和。

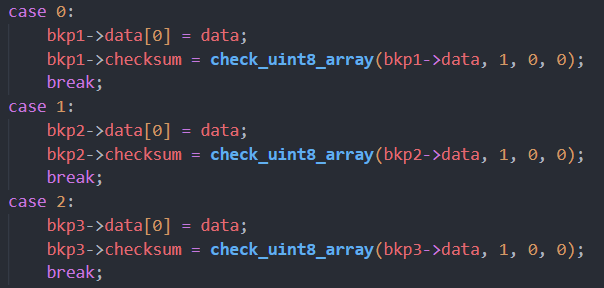


然后再将正确数据送回程序常用的全局变量中：



·备份更新

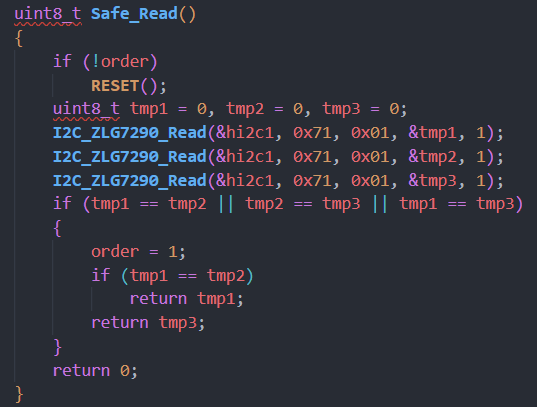
备份更新便是直接将全局变量写入备份中，并计算校验和。



在程序中，每次进行已备份全局变量的修改时，就要进行备份的更新操作。

## 输入滤波

程序需要读取按键输入，为了防止读入干扰数据，采取连续读取三次，当其中至少有两个相同数据才接收的方式。

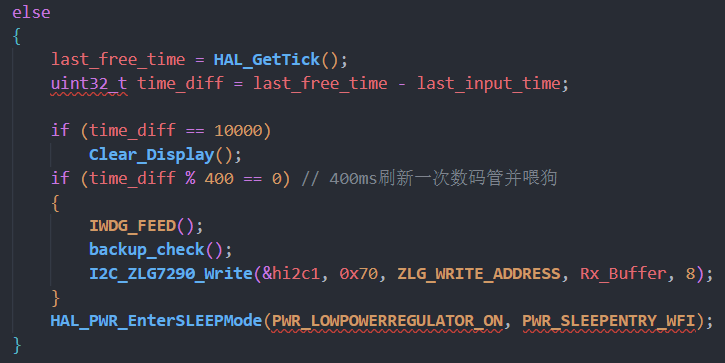


## 输出刷新与配置刷新

在用户不输入时，为防止数码管显示和内存出错，需要定期进行检查，先声明两个全局变量用于记录最后一次输入的时间和最后一次空闲的时间，分别在键盘中断发生时和空闲时读取当前时钟更新时间：

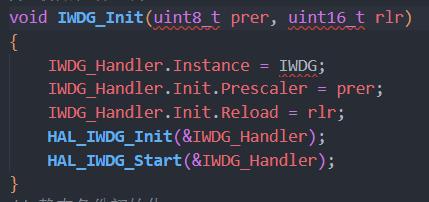


在main函数的else部分，计算最后一次输入到现在的时间，若超过10s，则将计算器还原；每隔400ms进行一次备份的检查和数码管显示的刷新：



## 看门狗

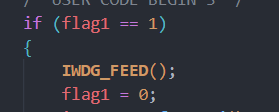
在程序发生故障卡死，或者被攻击使得程序跑飞时，需要看门狗自动重启纠正错误。看门狗命令在程序的中断中拥有最高的优先级，使用独立时钟，即使主时钟发生故障，它也仍然有效。看门狗初始化和启动函数如下：

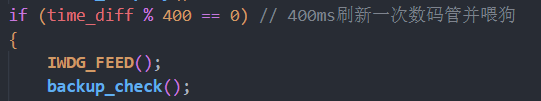


参数如下，根据计算公式，600ms进行一次看门狗检查，由于时钟周期可能会被温度影响，因此喂狗周期需设为400ms左右。



程序中有两处进行了喂狗（实在不知道怎么改成只有一处），分别为接收到输入时进行一次喂狗，在程序空闲时每隔400ms进行一次喂狗：



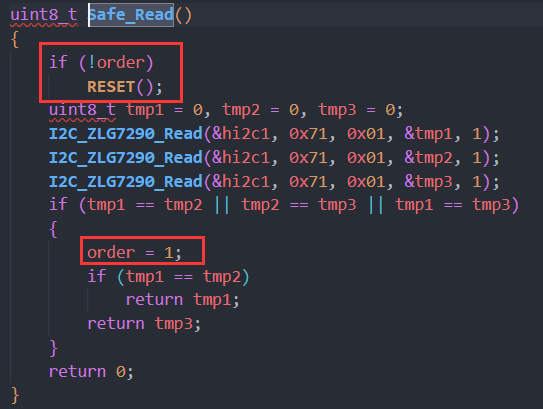


## 前序代码检查

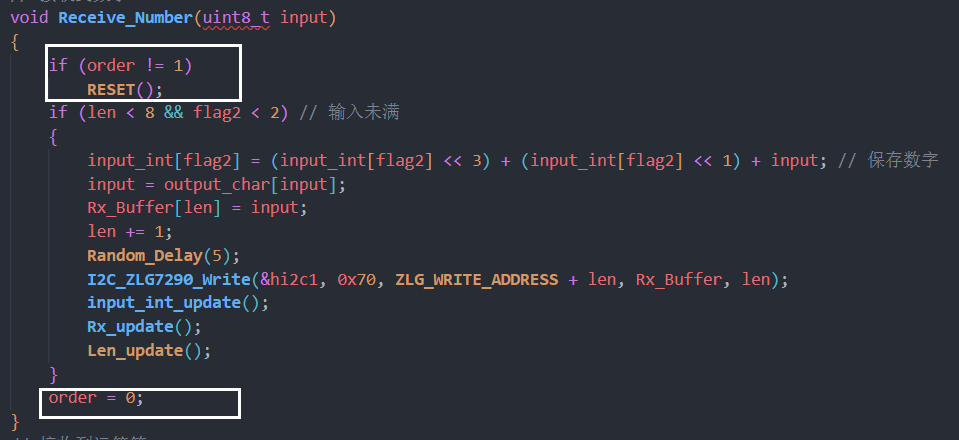
为了防止攻击者恶意攻击从而使得指令执行顺序出错，程序中还需要进行执行顺序的检查，我们使用了两个全局变量order和flag2，前者用于保证程序物理流程不出错，后者用于保证程序逻辑流程不出错。

·order

程序在每轮循环时只有两步，第一步是读取输入，第二步是输入处理。所以只需要在读取输入前判断order是否为0，读取后将order置1：



在每个数据处理函数前判断order是否为1，处理完成后将order置0：



·flag2

flag2是用于保证程序逻辑顺序不出错，flag2为0表示目前正在接收第一个参数，为1表示目前正在接收第二个参数，为2表示正在进行计算，为3表示计算结果合法。

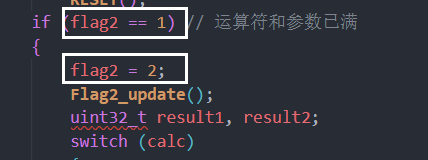
在处理数字时需要判断flag2是否为0或1：



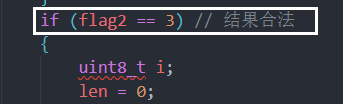
在处理运算符前，由于目前只能处理两个参数，需要判断flag2是否为0，并且接收运算符后要将flag2置1接收下一个参数：



在计算结果时，需要判断flag2是否为1，即已经接收过两个参数，并将flag2置2表示进入运算，当运算结果合法时，flag2自增1：



在flag2为3即结果合法时，才会进行结果的显示，否则会清空显示并还原计算器：

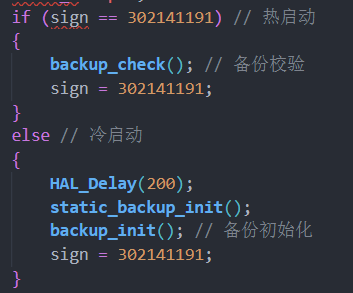


## 上电延时与冷热启动

当程序出现差错或遭到攻击导致重启时，不能够影响用户的使用体验， 要实现这样的功能就需要热启动。在热启动下，系统对某些数据不会进行从0开始的初始化，而是保留上次结束程序时的样子。首先需要在不初始化区域声明一个标志：



在标志为特定值时，表示程序需要热启动，此时只需要校验备份并将备份赋值给全局变量。在标志不为特定值时进行冷启动，延迟一段时间让硬件初始化，然后初始化所有备份。



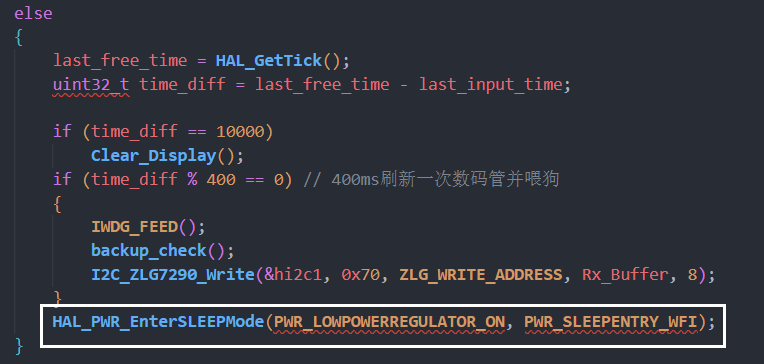
通过冷热启动的方式，在计算器某次计算过程中发生复位时，计算器能够保存所有已有数据，不会打断计算。

## 睡眠躲避干扰

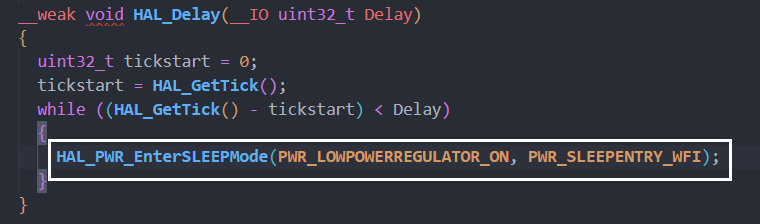
当程序在一段时间内不执行有效指令时，便可以设置让CPU进入睡眠模式，该模式不仅能降低CPU功耗，还能躲避干扰。睡眠模式的设置非常简单，stm32提供了多个CPU模式，分别是低功耗睡眠模式SLEEP、停止模式STOP以及正常运行。停止模式唤醒条件苛刻，选择睡眠模式即可。

进入睡眠模式使用函数为HAL\_PWR\_EnterSLEEPMode，它的两个参数分别为regulator寄存器是否打开和通过事件唤醒或通过中断唤醒。选择关闭regulator寄存器和中断唤醒即可，这样CPU会被时钟中断唤醒。

不输入时进入睡眠：



延时函数进行空while时睡眠：



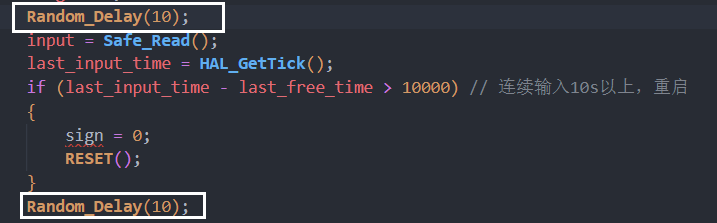
## 随机延时

嵌入式硬件设备容易遭受侧信道攻击。随机延时可以有效防御该攻击，模糊攻击者对于数据处理过程的推断。

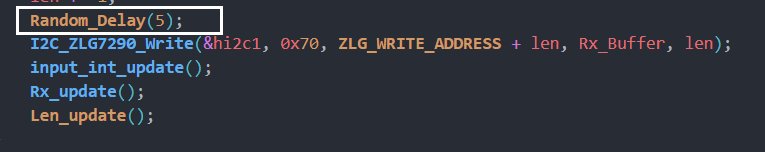
编写一条随机延迟指令：



将该指令插入读取键盘前后



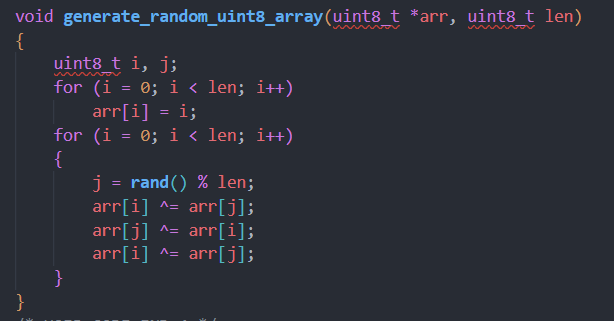
以及更新数码管显示前：



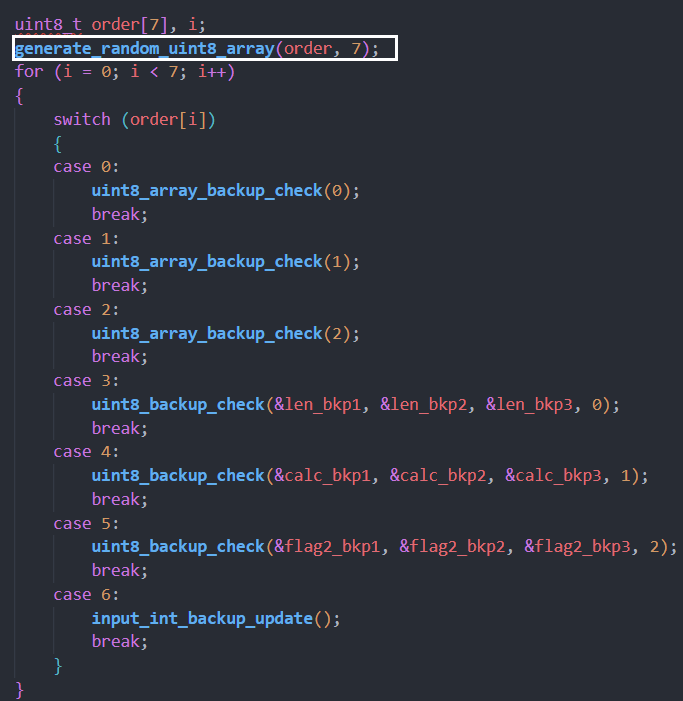
## 随机乱序执行

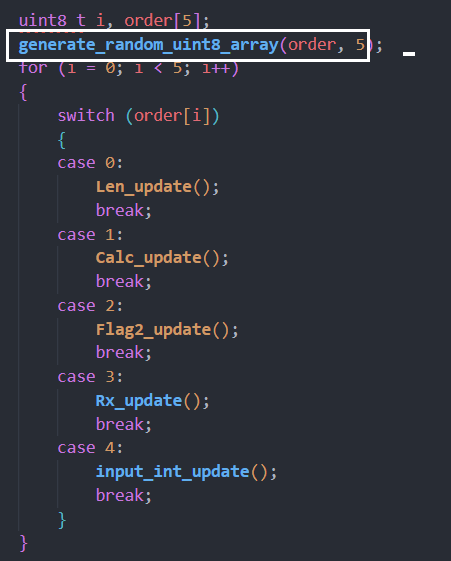
与随机延时一样，随机乱序执行同样是为了抵御侧信道攻击。主要实现方法为，将一些执行顺序可颠倒指令以随机顺序执行。这样的指令在备份初始化和校验中存在。

首先编写一个生成乱序数组的函数，它将一个完整的数组顺序打乱：



将随机乱序应用于备份初始化和校验处：

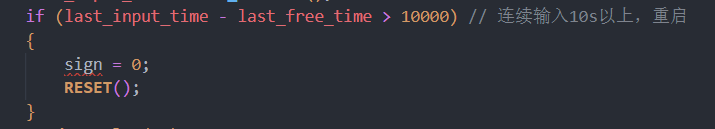




## 超时判错

当用户连续输入超过一定时间时，程序会将其视为发生了错误，这是就需要进行冷启动来恢复初始状态。由于之前已经生命过两个用于把保存时间的全局变量，再次使用即可。

当最后一次空闲的时间到现在超过10s时，进行冷启动：



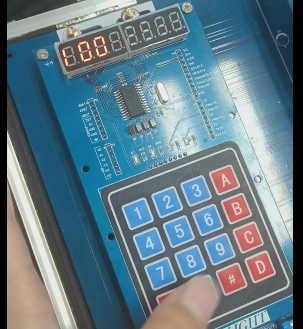
# 功能展示

1. 加法测试，输入123 A 123 \*， 得到结果246

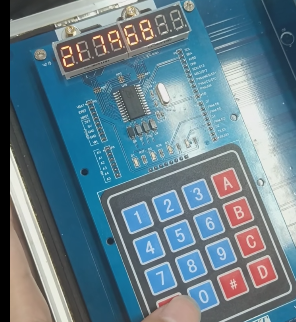
1. 减法测试，输入154 B 54 \*，得到结果100

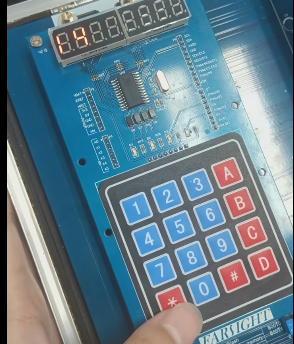


1. 乘法测试，输入456 C 478 \*，得到结果217968



1. 除法测试，输入 123 D 14 \*，得到结果8



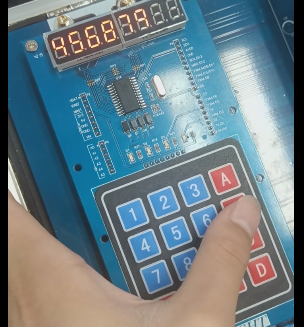
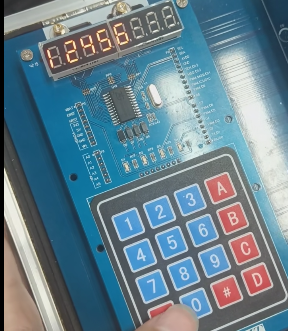
1. 其他测试

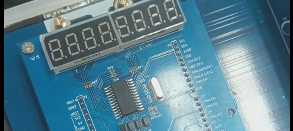
测试456 / 0，不合法所以不会显示结果



测试456879 \* 12456，超出长度限制，也不会显示结果



# 实验感想

最开始我准备编写一个功能更加丰富的嵌入式程序，但在真正编写的过程中可以说是问题不断。最开始我想把所有内容放入main函数，随后发现代码越写越多，原本以为可以用较少代码量写完计算器，于是便将各个模块拆分且削减的一些功能，只留下了计算器。

计算器的编写过程遇到了很多问题，首先是数码管的显示，以及各个按键对应的字节码，通过串口调试工具，这两个问题并不难解决。获取字节码后，我通过一个字节数组进行转换，转换为方便处理的有规律的操作码。随后便是参数切换的问题，刚开始准备在输入运算符后下一次输入数字时进行清屏，就和市面上的计算器一样，随后发现要实现这样的功能远不如直接在输入运算符后清屏方便。最后，就是运算和处理部分，也出现了许多问题：一是结果合法的判断上，一开始没什么问题，乘法的合法性使用最大值除以某个参数的方式判断防止溢出，后来发现这样的判断方式无法判断参数为0的情况，于是新增了某个参数为0时乘法结果也合法；二是结果显示上，由于需要将结果各个十进制位上的数取出，设置了一个条件为result != 0的循环，但是这样就无法显示结果为0的情况，于是给结果为0单独增加了一个if语句来显示。

在安全设计方面，我也遇到了很多问题。；数据备份上，我最开始将备份放在一个固定的内存，并将内存设置为不初始化，在老师指导下得知这只是在实验中可行，实际情况下我们并不清楚代码使用的内存，所以应该将备份放入块中，并设为不初始化。而且，我也意识到功能较复杂的弊端，在进行数据备份时，我发现我有好多数据必须备份，这些数据有不同的类型，所以不能用统一的结构体来表示，所以在数据备份上写了很多重复代码；看门狗方面，我的程序中有两处喂狗，但是实际上程序中应该值出现一次，这样攻击者命中喂狗的可能性最低，但是我并没有想到在我的程序中如何优化到只有一条；在输出刷新方面，一开始我只设置了空闲时每隔400ms刷新一次输出，但我没考虑到这段时间内存也是可能出错的，没有进行所有备份数据的检验。

经过这次嵌入式程序的编写和安全设计，我有不少收获。以前我没想过写一个程序要考虑那么多安全方面的因素，这让我意识到了系统安全可靠的重要性。在程序发生故障时，不仅要能做到自我修复，还要不影响用户的正常使用。对于外界的攻击，也要有一定的防御能力。我相信这样的思想对我以后得学习和工作会带来很大的帮助。