SH30F9/SB0 系列应用指南

-基于 ARM® Cortex™-M0+内核的 32 位高性能芯片

- Version 1.0

版本历史

版本号	修改记录	时间
V1.0	原始版本	2024-09

文档使用说明

本应用指南是 SH30F9010/30F9011/30F9810(以下简称 SH30F9/SB0)微控制器产品应用的指导性手册，主要包含有关如何使用 SH30F9/SB0 系列产品的基本信息和特别注意事项，包含功能模块典型工作模式的配置和使用方法，会以示例代码形式提供基本操作方法。

本应用指南不包含有关产品功能和技术特征的详细说明，包括模块内部结构、完整功能介绍、引脚数量和分配、电气特性、封装信息等，这些内容在产品数据手册中提供。

本应用指南示例代码使用直接寄存器操作方法，不使用封装的外设库函数操作方法，基于外设库进行产品开发请参考 SH30F9/SB0 系列 Demo 中的“SH30F9XX_SB0_STD_LIB_MANUAL. chm”说明文档。例程中用到的结构体在头文件“sh30f9xx_sb0.h”中定义。

目录

1. RAM 应用指南

1.1. 概述

SH30F9/SB0 系列芯片只有一块 SRAM。SRAM 位于 0x20000000 区域，主要用来存放程序变量，也可运行程序代码，带有写保护功能，可以保护在 SRAM 中的代码不被无意篡改。

1.2. 编程设置

SH30F9/SB0 系列芯片的 SRAM 写保护以 1K Bytes 为一个扇区，一个保护位对应一个扇区，由 SYSCFG 模块中的 SRAMLOCK 寄存器控制。

【注意】对写保护后的 SRAM 再进行写操作，会触发 Hardfault 异常。

【注意】将特定的程序放到 SRAM 中运行需要在程序中标识该程序段，并在 Linker 的配置文件中指定该标识段的地址到 SRAM 中。进一步信息请参考附录 1。

2. NVIC 中断配置应用指南

2.1. 概述

NVIC 是 Cortex-M0+的主要组成部分，它与 CPU 紧密结合，降低了中断延时，并允许新进中断可以得到高效处理。SH30F9/SB0 系列芯片除了 Cortex-M0+提供的异常外，还有 32 个可屏蔽中断，4 个可编程中断优先级（使用 2 位中断优先级）。

2.2. 编程设置

2.2.1. 中断配置基础

每一个中断都有一个中断号，是所有中断操作的身份，定义在芯片头文件中。中断向量表的起始地址由寄存器 VTOR 确定，复位时此寄存器为 0。中断向量表定义在 startup_sh30f9xx_sb0_keil.s 中，且为每个中断定义了一个默认的处理函数。当需要替代此处理函数时，只需要在代码中定义一个与中断向量表中名字相同的中断服务函数即可。无需对 startup_sh30f9xx_sb0_keil.s 进行任何修改，编译器会自动替换为自定义的函数。

2.2.2. 中断的使能与除能

在使能中断时首先要将模块内部的中断使能位置 1，然后再将 NVIC 中的中断使能位置 1。除能时只要其中任意一个除能就不会执行中断服务程序。另外特殊功能寄存器 PRIMASK 可以更方便的屏弊中断。该寄存器可通过 MSR，MRS 指令访问。

PRIMASK：中断屏蔽寄存器，只有 1 位。置 1 时屏蔽所有可屏蔽中断。

2.2.3. 中断的挂起与释放

通过中断挂起寄存器(SETPEND)和中断挂起清除寄存器(CLRPEND)可以对中断的状态进行控制。从而实现用程序触发某个中断的功能。

【注意】写中断挂起寄存器操作对已经挂起或已经被禁止的中断没有影响。

【注意】NVIC 中的 PEND 寄存器会自动置位清 0，因此在非特殊情况下无需对其修改。

2.2.4. 中断的优先级

SH30F9/SB0 系列芯片的外部中断都有一个对应的优先级寄存器，每个寄存器有 8 位，Cortex-M0 系列只能使用最高 2 位。

2.2.5. 中断/异常的响应序列

入栈：将 8 个寄存器入栈（xPSR,PC,LR,R12,R3,R2,R1,R0）

取向量：从向量表中找出对应的服务程序入口地址

更新寄存器: 选择堆栈指针 MSP/PSP，更新堆栈指针 SP，连接寄存器 LR，程序计数器 PC

SP:入栈中会把堆栈指针(PSP/MSP)更新到新位置。在执行服务例程后，将由 MSP 负责对堆栈的访问。

PSR: IPSR 位段（地处 PSR 的最低部分）会被更新为新响应的异常编号。

PC: 在向量取出完毕后，PC 将指向服务例程的入口地址。

LR: LR 的用法将被重新解释，其值也被更新成一种特殊的值，称为“EXC_RETURN”，并且在中断/异常返回时使用。

更新 NVIC 寄存器：中断挂起位被清除，活动位被置 1。

2.2.6. 中断/异常返回

当中断/异常服务程序执行完毕后需要做一个返回序列。有 3 种方法可以触发返回序列

BX <reg>当 LR 存储 EXC_RETURN 时，使用 BX LR 即可返回

POP {PC} / POP {…,PC}堆栈中存储的是 EXC_RETURN 且往 PC 出栈时。

LDR / LDM 把 PC 作为目标寄存器载入 EXC_RETURN 时。

返回序列

出栈：恢复之前压栈的寄存器

更新 NVIC 寄存器：中断活动位被硬件清除。对于外部中断，如果中断输入再次被置为有效，挂起位也将再次置位，新一次的中断响应序列也随之再次开始。

2.2.7. 嵌套的中断

CM0+内核的 NVIC 支持中断嵌套。需要配置中断优先级控制中断的嵌套。

【注意】主堆栈的容量要保持安全值，每嵌套一级至少需要 8 个字的空间。

【注意】相同的中断不允许重入。

2.2.8. 咬尾中断

如果挂起的中断的优先级比所有被压栈的中断的优先级都高，则处理器执行咬尾链接(Tail-chaining)。咬尾链接就是在退出 ISR 并进入另一个中断时，处理器略过 8 个寄存器的出栈和压

栈操作，因为它对堆栈的内容没有影响。因此能够在两个中断之间没有多余状态保存和恢复指令的情况下实现背对背处理。

2.2.9. 晚到中断

如果前一个 ISR 还没有进入执行阶段，并且迟来中断的优先级比前一个中断的优先级要高，则迟来中断能够抢占前一个中断。响应迟来中断使需执行新的取向量地址和 ISR 预取操作。迟来中断不保存状态，因为状态保存已经被最初的中断执行过了。

3. 电源、时钟和复位应用指南

3.1. 概述

电源、时钟和复位电路是工控类芯片设计的核心，也是可靠性和抗干扰能力的保证。

电源方面，支持 2.7V～5.5V 供电范围，是真正的宽电压供电 MCU，又增加了低电压保护（LVR）和掉电检测（BOD）常规保护电路。

时钟方面，支持 HSI、HSE、LSI 三种时钟源，其中 HSI 时钟在常温下达到 0.4%精度，在大部分控制应用场合中可以替代晶振作为高精度时钟源使用。

复位电路方面，支持上电复位、独立的引脚复位、低电压复位、看门狗复位以及内部软件复位，复位方式多样和完善。

3.2. 编程设置

3.2.1. 电源

【注意】电源上电会有个预热（稳定）时间，建议预留 18ms（15ms 基础上预留一定裕量）。从停机模式下唤醒或发生复位时重新运行，因为不涉及重新上电问题，可把电源预热压缩到 1ms 左右。

【注意】BOD 是个灵活的掉电检测电路，支持下降检测、上升检测、双向检测，还可查询当前电压与设置电压的高低状态，BOD 产生后可申请 BOD 中断，也可以申请 NMI 中断，后者由 IEN_BOD@SYSCFG_SAFR 控制位开启。

【注意】BOD 中断无法唤醒 STOP 模式，建议不要试图用 BOD 中断去唤醒 STOP。

【注意】为提高抗干扰能力，BOD 设计有去抖动功能，防止由于 VDD 上的干扰和毛刺引起 BOD 误发生。

【注意】LVR 有三档选择，LVR 设置要参考 VDD 电压，同时要比芯片最低工作电压高。比如 VDD=5V，可设置 LVR=4.1V，VDD=3.7V，可设置 LVR=3.1V。LVR 设计有回差和去抖动功能，防止由于 VDD 上的干扰和毛刺引起 LVR 误发生。

【注意】BOD 由内部寄存器开启和设定 BOD 门限电压，LVR 也是由内部寄存器开启和设定 LVR 电压。

3.2.2. 时钟

【注意】HSE 除了外接晶振外，还可以外灌时钟，从 XTAL1 引脚灌入，可以外灌方波或正弦波。

【注意】所有时钟源的切换都应等对应 READY 置位后才进行。

【注意】HCLK、PCLK0、PCLK1 都有最高频率限制，在配置总线时钟时需注意。

【注意】CSM 默认关闭，可以通过配置寄存器开启。CSM 发生后系统自动切换到内部时钟 8 分频运行，并申请 NMI 中断（由 IEN_CSM@SYSCFG_SAFR 控制位开启），在 NMI 中断中可以做一些系统保护工作。

【注意】发生 CSM 后，除了 HSECSMF 被置位外，RCC 的一些控制位不会清零，如 HSEON、LSION、SW、SWS 都保持原值，即此时系统时钟已经切换到 HSI/8 时钟，但 SW 仍保持原设置。

【注意】系统时钟选择 HSE，CSM 模块开启时，HSI 禁止关闭，否则 CSM 工作异常。

【注意】CSM 功能必须在 HSERDY 之后再开启。

【注意】发生 CSM 后，虽然系统时钟已自动切换到 HSI/8，但为了系统稳定必须将 AHB 总线分频系数设置为 8，并将时钟切换到 HSI，待失效时钟恢复（HSERDY 标志置起）后，清除 CSM 标志，此时系统时钟切回异常前时钟，再将 AHB 总线分频系数改回原设定值。

【注意】SysTick 默认使用 HCLK/8 作为时钟源，周期常数固定为 60000，在 HCLK=48MHz 时 SysTick 周期为 10ms，如果 HCLK 为其他主频，相应的周期也自动变化，计算方法是 60000*8/HCLK。

【注意】各模块的复位由软件置位，复位完成后硬件自动清零，启动复位后要求等待一个 HCLK 时钟再去读取复位后参数。

3.2.3. 复位

【注意】SH30F9/SB0 系列芯片有完善的上电复位电路，按键（引脚）复位不是必需的，保留按键（引脚）复位是为了灵活性和功能性考虑，比如把 SWD 调试接口全部复用为 GPIO 口后要恢复调试功能就需要按键（引脚）复位介入，具体在 SYSCFG 模块有详细介绍。

【注意】上电复位、低电压复位后有一个电源预热时间，按 18ms 计，而按键（引脚）复位、看门狗复位、软件复位则只需预留 1ms 左右预热时间。

4. GPIO 用户指南

4.1. 概述

4.1.1. GPIO 模式

GPIO 模式是 I/O 的基本工作模式，上电完成后，除调试接口（SWD port）外，所有 I/O 口处于全关模式(AF = 0)，若需要使用 GPIO 的功能，请将其相关 AF 设置为 1，不使用的 GPIO 口推荐设置 AF 为 0。

GPIO 模式有 3 种基本工作方式：GPIO 输入、GPIO 输出（推挽）、GPIO 输出（开漏）。

GPIO 输入：MODER=0b， PHDRx=0b/1b，对输入寄存器 IDR 的读访问可得到 I/O 口电平状态。

【注意】读操作才触发 I/O 口电平采样，其他时间 IDR 保持上次采样值。

GPIO 输出（推挽）：MODER=1b，OTYPE=0b，ODRVR_SINKx=0b/1b，对输出寄存器 ODR 的写操作将改变 I/O 口电平，写 1 激活 PMOS，输出高电平，写 0 激活 NMOS，输出低电平；

GPIO 输出（开漏）：MODER=1b，OTYPE=1b，ODRVR_SINKx=0b/1b，对输出寄存器 ODR 的写 0 可以激活 NMOS，输出低电平，写 1 无法激活 PMOS，无法输出高电平，处于高阻状态；

【注意】开漏方式下通过外部上拉获得高电平输出，但外部上拉不要超过芯片工作电压，极限情况下不要超过 VDD+0.3V。

【注意】输出方式下调节 ODRVR_SINKx 可改变灌电流的能力。其中 70mA 灌电流能力仅局部 8 个引脚具备，为 PB2～PB9/PC3~PC10。受限于芯片 GND 最大电流限制 200mA，应用中每次只能启动 1 路 200mA 灌电流，如果有多路只能以扫描方式分时启动。

4.1.2. 复用为数字外设

数字外设可通过 AF 寄存器选择映射的 I/O 口，SH30F9/SB0 系列芯片支持强大的引脚重映射，同一个外设可映射到不同位置的 I/O 口上，方便系统布线和功能组合。

数字外设会控制 I/O 口的输入输出，即某个 I/O 口映射为数字外设后无需用户配置 MODER 寄存器。除此之外 I/O 口的其他功能选项还需用户软件配置，如上拉、推挽/开漏、输出驱动能力。

【注意】数字外设的配置和 I/O 口配置并没有先后顺序的严格要求。一般而言，对输出口建议先配置外设再配置 I/O 口，对输入口或双向口先配置 I/O 口再配置外设。前者是因为有些外设在开启之前输出口会呈现高阻状态或一个电平，开启后又会呈现一个电平，如果预先映射好 I/O 口，电平变化可能会对外部电路有影响。后者则是对外设输入口而言，开启之前要求有一个确定性的电平，因此需预先配置好 I/O 口，避免电平变化引起外设误动作。

配置顺序可简单总结为：

输出类外设：先配置外设，开启外设，再配置 I/O 口上下拉、输出方式，最后 AF 切换到外设。

输入类外设：先配置 I/O 口上下拉、输出方式，再 AF 切换到外设，最后再配置和开启外设（如果外设有使能开关，可以在 I/O 配置时同步做外设配置，最后一步再开启外设）。

映射为数字外设时会断开 GPIO 输出（实际只断开 ODR 寄存器输出），但不会关闭 GPIO 输入，即通过 IDR 总是能够采样到 I/O 口电平。

通过“复用功能映射表”可以很方便确定外设与 I/O 口的映射。

【注意】SH30F9/SB0 系列芯片没有禁止重叠映射，比如 RXD0 可以同时映射到 PA4、PC3、PC4、PC5、PD0、PD1，一旦这样操作，这 6 个引脚的接收信号都将送入 RXD0，引起信号紊乱，应用中应避免这种配置。同理，TXD0 硬件上也允许这种重叠映射，也要由用户自觉避免。

4.1.3. 复用为模拟外设

映射为模拟外设时 GPIO 输入输出通道都被关闭。此时如果用户读 IDR 寄存器将会读到 0。

模拟外设配置 MODER、OTYPE、ODRVR 寄存器是没有意义的，但允许配置 PUPDR 寄存器，即可在 I/O 口上构造一个上拉。

【注意】模拟外设要求 I/O 口先切换到相关 AF，再打开模拟外设。

注：这点和数字外设不同，数字外设只是建议，模拟外设是必须。

4.1.4. 全关模式

全关模式会切断输入输出通道，可彻底消除 I/O 口在配置为输入悬空时存在的漏电问题，简单起见，应用时可以把不用的 I/O 口配置为全关。

上电复位期间，除 SWD 口外的所有 I/O 口都处于全关状态，复位完成后进入默认状态（GPIO 输入输出全关）。

4.1.5. 调试接口

两线 SW 调试接口（SWD 接口）：SWDIO、SWCLK 都可以复用为 GPIO 模式。

系统上电复位后默认作为调试接口，要复用为 GPIO 模式需通过 SWJCFG@SYSCFG_SAFR 寄存器修改。

【注意】此处 GPIO 模式包含数字外设模式，统指“非调试接口”模式，可以作为数字外设映射口。

【注意】振荡器接口 XTAL1/XTAL2 复用请参考 OSCCFG@SYSCFG_SAFR 寄存器定义，上电默认为输入输出全关状态。

【注意】SYSCFG_SAFR 寄存器中 SWJCFG、OSCCFG 二个控制位都是在 reset 后只能设置一次，程序中一旦置位无法更改。

4.1.6. 其他注意事项

【注意】I/O 的配置操作有对应的锁定/解锁功能，锁定后配置信息无法被修改。

```
GPIOB ->LCKR.V32 = 0x5AA5FFFF;          // lock PB config
GPIOB ->LCKR.V32 = 0x5AA50000;          // unlock PB config
```

【注意】端口的置位和清零有两种方式：BSRR 和 ODR，其中 ODR 的置位和清零操作就是通过 BSRR 实现的。当 BSRR 对某一位的置位和清零同时有效时，置位具有高优先级。

```
GPIOB->BSRR.BIT.BS = 0x8000;        // set PB15 = 1
GPIOB->BSRR.BIT.BR = 0x8000;        // set PB15 = 0
GPIOB->ODR |= 0x8000;               // set PB15 = 1
GPIOB->ODR &= 0x7FFF;               // set PB15 = 0
```

【注意】使用 GPIOx->ODR 寄存器时，应注意在使用前关中断，设置完成后再打开中断，建议客户使用 GPIOx->BSRR 寄存器对 IO 进行置位或清零操作

【注意】TTL 电平输入功能。该功能主要针对外部以 TTL 电平工作的端口与本芯片连接时的电平匹配问题。 TTL 电平输入选项改变输入高电压和低电压门限（VIH3、VIL3）。

【注意】TTL 电平输入功能在 STOP 模式下仍然有效。

5. Flash&EEPROM 用户指南

5.1. 概述

SH30F9/SB0 系列芯片内置最大 256K 的可编程 Flash 主程序存储区（Main Program Memory Block），详细信息请参考规格书中产品信息章节，每个扇区 1024 字节， 每个扇区都可单独进行擦除。主存储区进行擦除操作后，每 16 位或 32 位仅可以被编程一次，即被编程的 16 位或 32 位数据的各个位如果不是 0，就不可以再次被烧写，否则相应错误标志置 1。。

4KB 的内置类 EEPROM 存储区，可存放数据，每个扇区 1024 字节，每个扇区都可单独擦除。

1KB 的 OTP 区，用于存放出厂初始化数据，需要注意的是 OTP 区数据一旦被写入就无法被擦除。

Flash 中的主程序区可以通过设置读保护防止被非法的读出；同样可以对 Flash 存储区的各扇区设置写保护，防止在程序跑飞的情况下不被意外地改变。写保护的基本单位是 8 个扇区为一个保护单元。

5.2. 编程设置

Flash、EEPROM、OTP 区的操作流程基本一致，大致分为以下几个步骤：

- 解锁对应的操作寄存器
- 设置 Flash 操作定时器
- Flash(E2\OTP)擦除烧写配置
- 锁定对应的操作寄存器

【注意】在对 Flash(E2\OTP)存储器进行擦除编程操作之前，请将 WDT 喂狗，LVR 功能关闭，避免在操作过程中发生复位，同时需要将总中断关闭，操作完成后再将总中断打开。

5.2.1. Flash 控制寄存器解锁

系统复位后，Flash 存储器操作寄存器进入锁定状态，所以对 Flash 进行擦除编程操作之前需要对 Flash 存储器操作寄存器进行解锁。不同 Flash 存储区具有不同的解锁寄存器，对不同区域操作需要解锁相应的寄存器。对 Flash 解锁需要对相应解锁寄存器依次写入两个解锁值，第一个解锁值是初始解锁值，第二个解锁值是单次或多次操作解锁值；第一个键值写入错误或第二个键值写入错误，都不能对 Flash 相应区域进行解锁，并会产生一个 HardFault；任意一个区域处于解锁状态，再对解锁寄存器进行解锁，也会产生一个 HardFault。对解锁 Flash 操作也分为两种情况，一种是解锁后可对 Flash 进行单次操作，另一种是解锁后，可对 Flash 进行多次操作。如果 Flash 存储器操作寄存器处于解锁状态，可通过控制 FLASH_CR，对 Flash 各存储区进行擦除、编程操作。

各解锁寄存器解锁值及说明

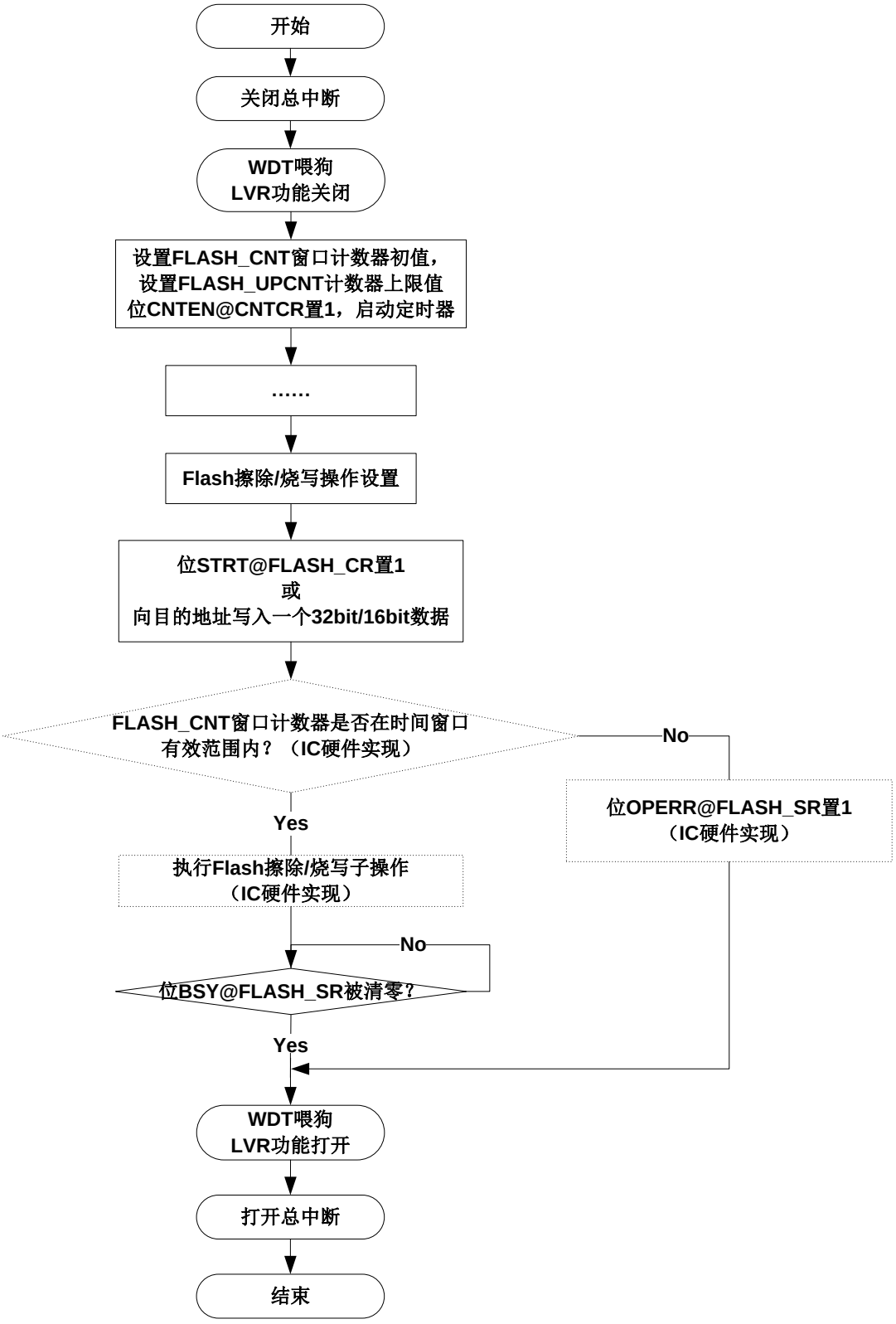
寄存器	功能	Flash 初始解锁值	Flash 单次操作解锁值	Flash 多次操作解锁值/锁定值	说明
FLASH_MKYR	解锁主程序区	0x8ACE0246	0xC3C3C3C3	0xB4B4B4B4	过运行在 RAM 中的程序，主程序区中的程序，Boot 区的程序或者使用调试工具直接访问
FLASH_E2KYR	解锁特殊信息区的类 EEPROM 区	0x9BDF1357	0xC3C3C3C3	0xB4B4B4B4	通过运行在 RAM 中的程序，主程序区中的程

					序，Boot 区的程序或者使用调试工具直接访问
FLASH_IKYR	解锁特殊信息区的代码保护区（Protect Block）、客户信息区（Customer Block）和 OTP 区	0xABCD5678	0xC3C3C3C3	0xB4B4B4B4	代码保护区、客户信息区和 OTP 区可以通过运行在 RAM 中的程序，主程序区中的程序，Boot 区的程序或者使用调试工具直接访问 Flash 操作寄存器来实现；而 Boot 区只能通过调试工具直接访问 Flash 操作寄存器实现。

5.2.2. Flash 操作定时器功能

Flash 控制模块具有一个操作定时器，Flash 操作定时器的计数值是以系统时钟为时钟源，Flash 操作定时器的计数值必须小于 Flash 操作定时器上限值，并且大于零，才可以对 Flash 存储区进行操作，或者向目的地址写入一个 32bit/16bit 数据。如果启动一次 Flash 操作，就是说位 STRT@FLASH_CR 置 1 或者启动对 Flash 的写操作时，Flash 操作定时器计数值不在有效的时间窗口内，Flash 操作不被执行，且位 PGWERR@FLASH_SR 置 1。Flash 操作定时器的具体用法可参考下图。

需要注意的是，CNTEN@FLASH_CNTCR 位控制的是操作定时器是否开始递减计数，当 CNTEN@FLASH_CNTCR 位置 1，启动定时器递减计数；当 CNTEN@FLASH_CNTCR 位为 0，Flash 操作定时器计数不变，因此，只要保证 Flash 操作定时器计数值不大于 Flash 操作定时器上限值，Flash 操作就可正常执行。



Flash 操作定时器使用流程图

5.2.3. Flash(E2\OTP)擦除烧写

对 Flash、EEPROM、OTP 区进行操作，在 Flash 控制器解锁、Flash 操作定时器设置后，就需要对 Flash 控制寄存器进行配置，需要配置的选项包括：

- Flash 操作命令字：CMD@FLASH_CR
  0xE619：主程序区扇区擦除(MSE)
  0x6E91：主程序区存储单元编程(MPG)
  0xB44B：类 EEPROM 编程(E2PG)
  0x4BB4：类 EEPROM 按扇区擦除（E2SE）
  0xF00F：特殊信息区 OTP 区编程（OPG）
- 烧写操作位宽选择位：PSIZE@FLASH_CR
  0：32 位同时烧写
  1：16 位同时烧写
- 扇区擦除扇区选择位:：SNB@FLASH_CR1
  00000000：扇区 0
  00000001：扇区 1
  00000010：扇区 2
  ……..
  11111110：扇区 254
  11111111：扇区 255

注：该位仅当主程序区扇区擦除或类 EEPROM 区按扇区擦除时才有效，主程序区按扇区擦除时，每个扇区大小是 1024 字节；类 EEPROM 区按扇区擦除时，每个扇区大小是 1024 字节。

- 开始标志位：STRT@FLASH_CR

当操作命令字是擦除时，当该位为'1'时将触发一次操作；当操作命令字是编程时，向目的地址写入一个 32 位字/16 位半字时将触发一次操作。该位只可由软件置为'1'并在 BSY@FLASH_SR 变为'1'时清为'0'。当 BSY@FLASH_SR 被清'0'，表示操作结束，此时需要查询 FLASH_SR 寄存器中的各状态位，判断操作是否正常执行，如果出错，相应的错误标志位将被置 1。

5.2.4. Flash 控制寄存器锁定

如果 Flash 控制寄存器的解锁方式是允许多次操作的话，那在完成 Flash 相关操作，务必将对应的 Flash 控制寄存器重新锁定，防止 Flash 被误操作。

5.3. 应用实例 Flash(E2\OTP)擦除烧写流程图

5.3.1. 主程序区整体擦除

Flash 操作提供了整片擦除功能，可以擦除主存储块区的内容。具体步骤如下：

（1）关闭总中断；

（2）将 WDT 喂狗，LVR 功能关闭，避免在操作过程中发生复位；

（3）FLASH_MKYR 依次写入 Flash 解锁值和单次操作解锁值，确保 Flash 主程序区不处于锁定状态；

（4）检查位 BSY@FLASH_SR 来判定 Flash 存储器是否处于运行状态；

（5）当位 BSY@FLASH_SR 为 0 时，写 Flash 主程序区整体擦除命令字到 FLASH_CR 寄存器；

（6）通过将位 STRT@FLASH_CR 置 1 来启动整片擦除操作；

（7）检查位 BSY@FLASH_SR 来判断擦除指令是否执行完毕；

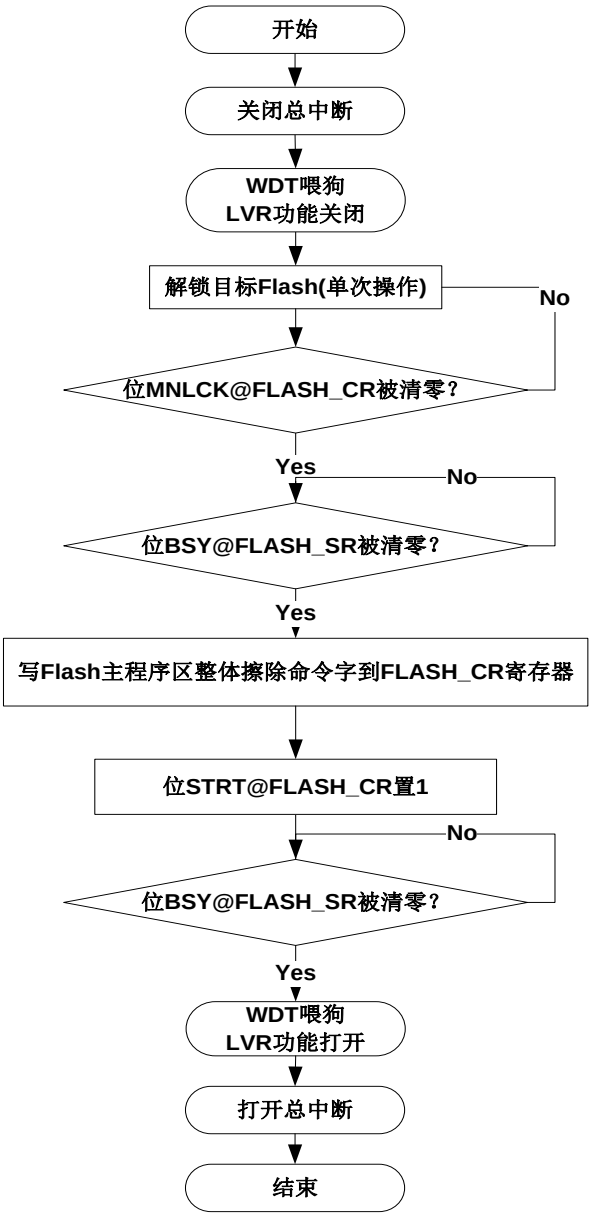
（8）当位 BSY@FLASH_SR 为 0 时，操作完成；

（9）将 WDT 喂狗，LVR 功能打开；

（10）打开总中断。

位 EOP@FLASH_SR 预示操作的结束，该位置 1，表示操作执行完毕。所有 Flash 数据擦除后被复位为 0x0000 0000。

流程图如下：



主程序区整体擦除

注意：在 Flash 主程序区运行程序执行整体擦除之前，需要将主程序区整体擦除保护字节（Addr: 0x0FFF8020）烧写为 0x01，才可以执行擦除操作。

5.3.2. 主程序区扇区擦除

Flash 存储器的每扇区都可以被独立擦除，不影响其它扇区内容。Flash 操作擦除扇区步骤如下：

（A）单独擦除某一扇区

（1）关闭总中断；

（2）将 WDT 喂狗，LVR 功能关闭，避免在操作过程中发生复位；

（3）FLASH_MKYR 依次写入 Flash 解锁值和单次操作解锁值，确保 Flash 主程序区不处于锁定状态；

（4）检查位 BSY@FLASH_SR 来判定没有 Flash 存储器是否处于运行状态；

（5）当位 BSY@FLASH_SR 为 0 时，写主程序区扇区擦除命令字和待擦除扇区号码分别到 FLASH_CR 和 FLASH_CR1 寄存器；

（6）通过将位 STRT@FLASH_CR 置 1 来启动扇区擦除操作；

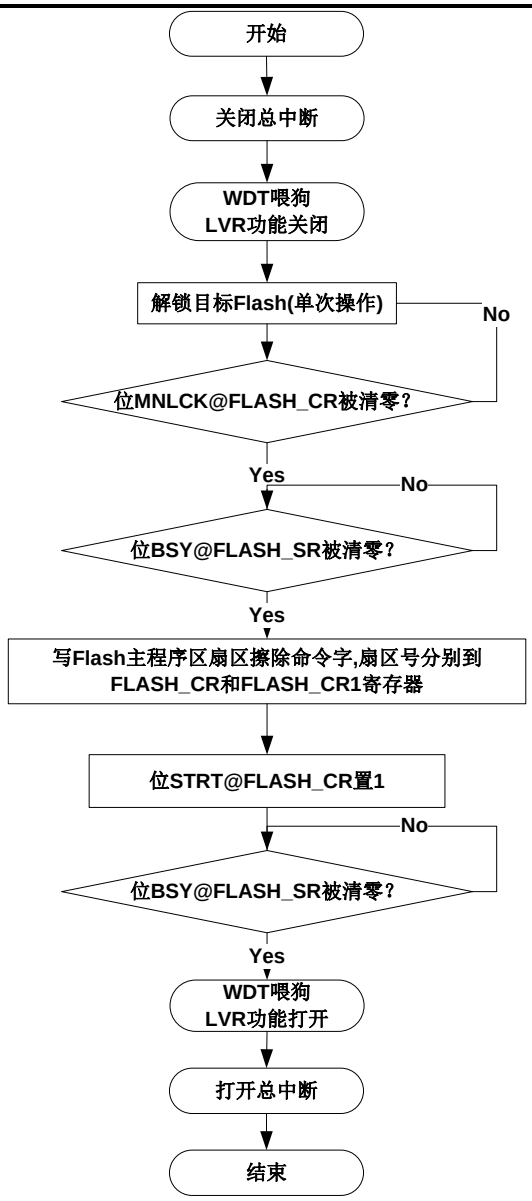
（7）检查位位 BSY@FLASH_SR 来判断擦除指令是否执行完毕；

（8）当位 BSY@FLASH_SR 为 0 时，操作完成；

（9）将 WDT 喂狗，LVR 功能打开；

（10）打开总中断。

流程图如下：



主程序区按扇区擦除（单一扇区）

（B）擦除多个扇区

（1）关闭总中断；

（2）将 WDT 喂狗，LVR 功能关闭，避免在操作过程中发生复位；

（3）FLASH_MKYR 依次写入 Flash 解锁值和多次操作解锁值，确保 Flash 主程序区不处于锁定状态；

（4）检查位 BSY@FLASH_SR 来判定没有 Flash 存储器是否处于运行状态；

（5）当位 BSY@FLASH_SR 为 0 时，写主程序区扇区擦除命令字和扇区代码分别到 FLASH_CR 和 FLASH_CR1 寄存

器；

（6）通过将位 STRT@FLASH_CR 置 1 来启动扇区擦除操作；

（7）检查位 BSY@FLASH_SR 来判断擦除指令是否执行完毕；

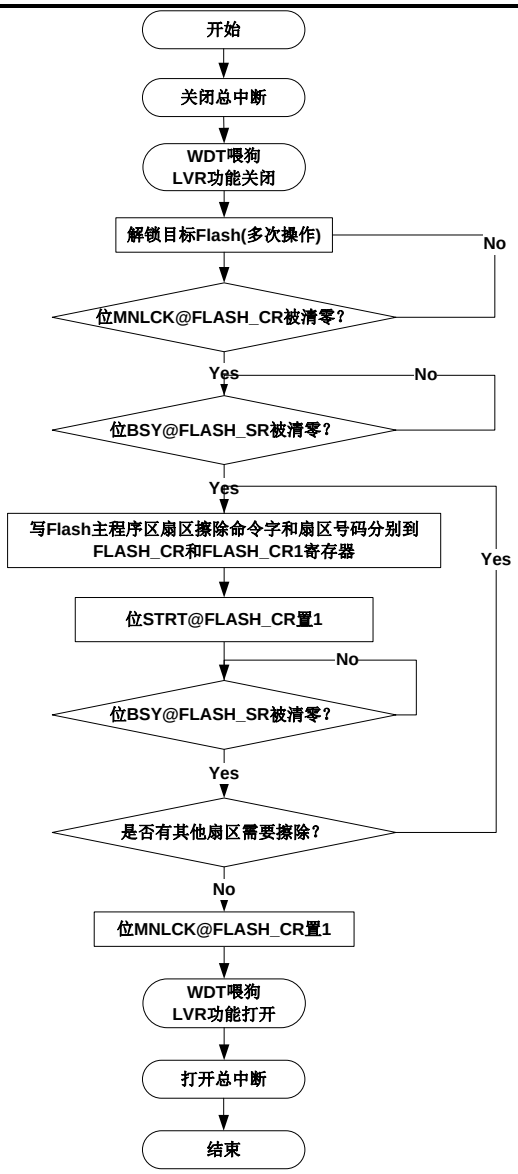
（8）重复（5）-（7），擦除其它扇区；

（9）位 MNLCK@FLASH_CR 置 1，操作完成；

（10）将 WDT 喂狗，LVR 功能打开；

（11）打开总中断。

操作流程图如下图：



主程序区按扇区擦除（多个扇区）

当目标擦除扇区被用来取指令或访问数据时，相应的擦除操作无效，但 Flash 操作将不提供任何通知，因此确保目标擦除扇区地址的正确性是很有必要的。位 EOP@FLASH_SR 预示操作的结束。

5.3.3. 主程序区编程

Flash 操作提供了一个 32 位字/16 位半字编程功能，用来修改 Flash 主存储块内容。对 Flash 主程序区编程每次可以写入 32 位/16 位，地址必需按 32 位/16 位对齐，若未对齐，相应操作无效，相关错误标志位置 1。当 FLASH_CR 寄存器操作命令字设置为主程序区编程时，在一个 Flash 地址写入一个字或半字将启动一次编程。在编程过程中(位 BSY@FLASH_SR 位为’1’)，任何读写 Flash 的操作都会使 CPU 暂停，直到此次 Flash 编程结束。

下面的步骤显示了字编程操作寄存器过程。

（A）单字或半字编程

（1）关闭总中断；

（2）将 WDT 喂狗，LVR 功能关闭，避免在操作过程中发生复位；

（3）FLASH_MKYR 依次写入 Flash 解锁值和单次操作解锁值，确保 Flash 主程序区不处于锁定状态；

（4）检查位 BSY@FLASH_SR 来判定没有 Flash 存储器是否处于运行状态；

（5）当位 BSY@FLASH_SR 为 0 时，写 Flash 主程序区编程操作命令字到 FLASH_CR 寄存器；

（6）向目的地址写入一个 32 位字/16 位半字；

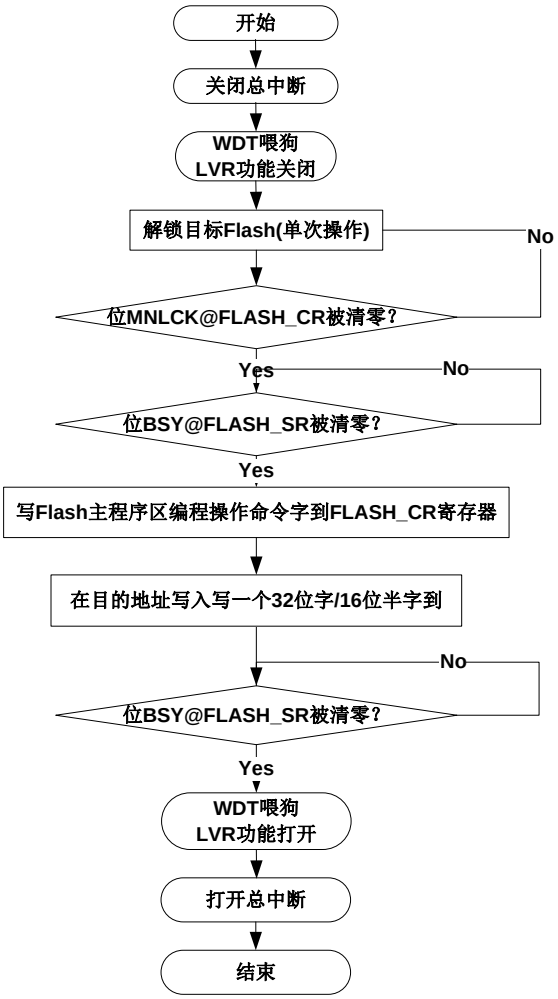
（7）检查位 BSY@FLASH_SR 来判断编程指令是否执行完毕；

（8）当位 BSY@FLASH_SR 为 0 时，操作完成；

（9）将 WDT 喂狗，LVR 功能打开；

（10）打开总中断。

操作流程图如下图：



主程序区编程（单字或半字）

（B）连续写入多字编程

（1）关闭总中断

（2）将 WDT 喂狗，LVR 功能关闭，避免在操作过程中发生复位；

（3）FLASH_MKYR 依次写入 Flash 解锁值和多次操作解锁值，确保 Flash 主程序区不处于锁定状态；

（4）检查位 BSY@FLASH_SR 来判定没有 Flash 存储器是否处于运行状态；

（5）写 Flash 主程序区编程操作命令字到 FLASH_CR 寄存器；

（6）向目的地址写入一个 32 位字/16 位半字；

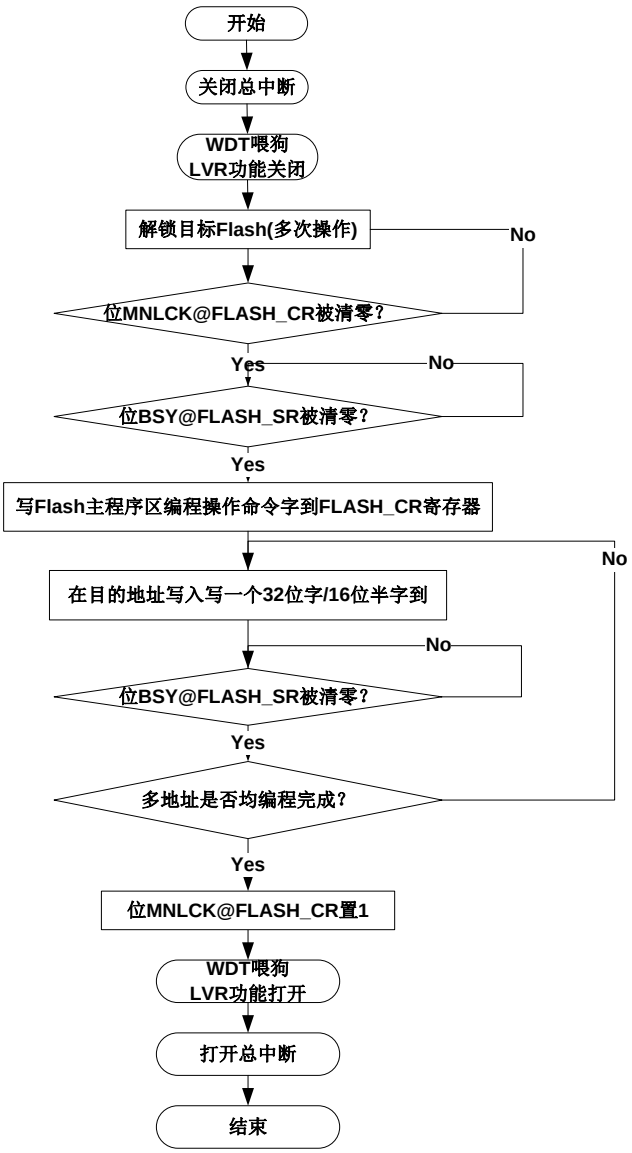
（7）检查位 BSY@FLASH_SR 来判断编程指令是否执行完毕；

（8）重复（6）-（7），直到多地址都编程结束；

（9）MNLCK@FLASH_CR 位置 1，操作完成；

（10）将 WDT 喂狗，LVR 功能打开；

（11）打开总中断。



主程序区编程（连续写入多字编程）

对 Flash 存储器，每次可以写入 16 位或 32 位数据。主存储区擦除操作后，每 16 位或 32 位仅可以被编程一次，即被编程的 16 位或 32 位数据中有一个位如果不是 0，就不可以再次被烧写，否则编程错误标志位 PGERR@FLASH_SR 被置 1。

如果 Flash 主程序区的某一扇区已经被写保护位保护，但仍对该扇区进行写操作，则写保护错误标志位 WRPRTERR @FLASH_SR 被置 1。

5.3.4. 类 EEPROM 存储区擦除编程

类 EEPROM 存储区擦除编程与主程序区按扇区擦除、编程相类似，不同的是 Flash 解锁寄存器不同，写入 FLASH_CR 寄存器的操作命令字不同，锁定存储区的标志位不同。

注意事项：在对类 EEPROM 存储区进行擦除编程操作之前，请将 WDT 喂狗，LVR 功能关闭，避免在操作过程中发生复位。

5.3.5. OTP 区编程

OTP 区无法被擦除，OTP 区的编程与主程序区编程相类似，不同的是 Flash 解锁寄存器不同，写入 FLASH_CR 寄存器的操作命令字不同，锁定存储区的标志位不同。

注意事项：在对 OTP 区进行擦除编程操作之前，请将 WDT 喂狗，LVR 功能关闭，避免在操作过程中发生复位。

5.4. 烧写/擦除抗干扰措施

由于程序代码中包含主程序区或 E2PROM 区等区域的擦除、编写功能的程序，当系统受到强干扰时，擦除编写操作就具有一定的风险性。比如在做 ESD/EFT 试验时，程序可能会跑飞，如果直接从外部跑飞到擦除编写代码处，可能会将非程序意图的改写或擦除，导致系统运行异常。

鉴于此原因，建议对 Flash 主程序区、EEPROM 区或其他区域操作时增加抗干扰措施：

(1) 由于只有当开始标志位 STRT@FLASH_CR 置 1，才会启动或等待 Flash 的擦除/烧写，所以在操作起始的处增加软件标志，然后在开始标志位 STRT@FLASH_CR 置 1 前检查该软件标志是否匹配，如果标志位不匹配，操作直接退出，因此不会启动烧写或擦除动作。

(2) 要充分利用 Flash 操作定时器的功能，可以在操作起始处启动 Flash 操作定时器，计算配置好允许操作的窗口时间，使得程序正常运行到执行擦除或编写操作时，Flash 操作定时器计数值刚好在可执行的窗口范围内，这样可以避免因程序跑飞导致的误操作。

(3) 尽量避免带参数的主程序区或 EEPROM 区烧写擦除子函数调用。因为程序跑飞时，参数是随机值，容易误操作。所以建议主程序区或 EEPROM 区擦除子函数，不带参数，函数中对地址赋常值，擦除某固定扇区。

(4) 对 Flash 主程序区、EEPROM 区或其他区域操作时，将对 Flash 解锁操作放到擦、写函数外面。

【注意】对 Flash（主程序区、EEPROM 区、OTP 区或其他区域）写操作时，只能按字（32 位）或半字（16bit）进行操作。

6. 代码和数据检测（CRC）应用指南

，

6.1. 概述

为方便数据校验，SH30F9/SB0 系列芯片提供了 CRC 模块，用来计算循环冗余码。此模块提供 4 种生成多项式及多种数据前后期处理以满足对 CRC 算法的不同的需求。

6.2. 编程设置

6.2.1. 生成多项式

支持 4 种通用的生成多项式：CRC-32 位（0x04C11DB7）、CRC-16 位（0x8005）、CRC-CCITT（0x1021)、CRC-8（0x7）。分别产生 32 位，16 位，8 位的 CRC 码。

6.2.2. 输入数据处理

支持将输入的数据进行格式转换后再进行 CRC 运算。支持的转换方式有按位取反、字节顺序逆向、字节内位顺序逆向。

6.2.3. 结果处理

支持将 CRC 的运算结果进行格式转换。支持的转换方式有按位取反、字节顺序逆向、字节内位顺序逆向。

6.2.4. 运算触发

将初始值通过 reload 控制位导入到内部运算寄存器后，只要往 DR 寄存器写值就会触发一次 CRC 运算。

6.2.5. 结果获取

读取 DR 寄存器，就得到 CRC 的运算结果，无需特殊等待。如果运算未结束，读取指令不会执行结束。

【注意】在开始连续运算前需要先 Reload 初始值。否则内部寄存器的初始值未知，会影响计算结果。同理，在连续运算过程中不能 Reload 初始值，否则会改变中间结果从而影响最终的结果

【注意】Reload 功能是将初始寄存器直接载入到内部寄存器，不会经过任何格式转换。当用分块分时计算某区域的 CRC 值时，如果此时对输出结果有格式转换处理，那么得到的结果就不是内部寄存器的值。所以在进行下一块的运算时就不能直接将上次得到的结果直接载入到内部寄存器，而需要经过格式转换的逆运算得到正确的中间结果后再载入到内部寄存器。简单的做法是：前面的几块先不设结果格式处理，待运算到最后一块时再设置结果格式处理。

【注意】往 DR 寄存器写入数据进行计算时，支持 8 位，16 位，32 位的写入。但要求写入/读取时寄存器地址必须按字对齐(4 字节对齐，即必须是 DR 寄存器的首地址)。否则结果不定。

【注意】当生成多项式为 32 位时，在对输入数据不作格式转换的前提下，一次写入 32 位数据进行计算与分 4 次写入 8 位（低字节在前）的结果是一样的。

【注意】当生成多项式为 8 位时，如果一次写入 1 个字，只计算一个字节，高位忽略。同样，当生成多项式为 16 位时，如果一次写入 1 个字，只计算一个半字，高位忽略。

7. 看门狗定时器应用指南

7.1. 独立看门狗定时器（HWDT）

7.1.1. HWDT 概述

SH30F9/SB0 系列芯片的独立看门狗具有一个 16 位的递减计数器，使用内建 128kHzRC 作为时钟源。

独立看门狗正常模式下无法关闭。

HWDT 可以工作在停机模式下，因此可以作为唤醒定时器。通过 HWDTEN@HWDT_CR 选择停机时开启或关闭看门狗（默认停机模式下开启）。

上电开启情况下，看门狗工作在较长溢出时间下，溢出时间大约为 2048ms。

7.1.2. HWDT 编程设置

在定时器溢出之前，通过对喂狗寄存器 HWDT_CLR 写入 0xAAAA 更新计数器以重新开始计数，避免产生溢出。

定时器溢出后将复位芯片，并置位 HWDTRSTF@RCC_RSTSTR 标志。

7.1.3. HWDT 应用实例

```
//配置独立看门狗溢出时间，T= 2.048s
HWDT->CR.V32 = ((0X5AA5<<HWDT_CR_LOCK_Pos)|0x00);
HWDT->CLR =0xAAAA; //喂狗
```

7.2. 窗口看门狗定时器（WWDT）

7.2.1. WWDT 概述

窗口看门狗通常被用来监测，由外部干扰或不可预见的逻辑条件造成的应用程序背离正常的运行序列而产生的软件故障。窗口看门狗是一个 8 位递减计数器，喂狗区间为一个窗口区，早于窗口喂狗区喂狗将产生超前异常事件，晚于窗口喂狗区将产生延迟异常事件，两者都会引起复位。另外，在计数器到达窗口喂狗区下边界时为最迟喂狗时刻，可申请 WWDT 中断，可作为窗口喂狗的最后补救时刻。

7.2.2. WWDT 编程设置

如果 RL 表示窗口看门狗的重载值，WT 表示窗口看门狗的喂狗上限值。如设置 RL>=WT 则是正常窗口监控功能，带有超前异常监视，即窗口看门狗必须计数器递减到喂狗上限值以下，才可以喂狗，否则会产生复位。如设置 RL<WT，则没有超前异常监控，只能监控延迟异常。如果延迟喂狗中断使能，在计数器到达窗口喂狗区下边界时为最迟喂狗时刻，可申请 WWDT 中断，可作为窗口喂狗的最

后补救时刻；若延迟喂狗中断禁能，在计数器到达窗口喂狗区下边界时直接发现复位。所以喂狗时间及窗口看门狗的设置务必根据程序做比较准确的计算。

注：如果设置 RL=WT=0xFF，则窗口最大，无超前监控，功能类似普通看门狗。

7.2.3. WWDT 应用实例

```
//配置窗口看门狗溢出时间
WWDT->CR.V32 = ((0x5AA5 << WWDT_CR_LOCK_Pos) |//寄存器解锁位
                (1 << WWDT_CR_WWDTON_Pos ) |   //启动 WWDT
                (1 << WWDT_CR_WWDTIE_Pos ) |//使能窗口看门狗最迟喂狗中断
                (WWDT_Prescaler_32 << WWDT_CR_WWDTPR_Pos) |//时钟分频
                (0xFF<< WWDT_CR_WWDTRLR_Pos) );              //重载值
//配置窗口看门狗上边限值
WWDT->WTR.V32 = ((0x5AA5 << WWDT_WTR_LOCK_Pos)   |      //寄存器解锁值
                 (0x7F<< WWDT_WTR_WWDTWTR_Pos) );     //喂狗窗口上限值

……（用户代码）
WWDT->CLR =0x5555; //喂狗
```

8. 外部中断（EXTI）应用指南

8.1. 概述

外部中断是由 16 个产生事件/中断请求的边沿检测器组成，可用于检测外部输入的边沿信号和电平信号。芯片上所有的 GPIO 引脚都可以配置为外部中断线，每个中断线可以独立的配置输入类型，也可以独立的被屏蔽。

8.2. 编程设置

8.2.1. 边沿触发模式选择

FTSR 和 RTSR 寄存器是选择相应外部中断线是否能被外部信号触发的，比如 RTSR 的 bit0 为 1，表示外部中断线 EXTI0 可以被外部的高电平/上升沿触发，其他位与此类似。一旦 FTSR 和 RTSR 相应位被置 1，只要外部有触发信号，PR 寄存器的相应位就会置 1，即使 IMR/EMR/DMR 等使能位为 0。

8.2.2. 软件触发中断

寄存器 SWIER 是软件触发寄存器，对 SWIER 的相应位写 1，就相当于相应的外部中断线上有一个外部信号触发。常用于外部没有触发信号，但程序中又需要进入外部中断函数去执行。此位写 1 后会自动清零，软件无需操作。

8.2.3. 外部中断线选择

每个 GPIO 引脚都可以配置为外部中断线，因此必然是多个 GPIO 引脚共用一个外部中断线，比如 GPIOA/GPIOB/GPIOC/GPIOD 的 PIN0 连接的都是 EXTI0，依此类推，EXTI0~15 分别与不同端口的 PIN0~15 连接，而具体的 EXTI 线选择哪个 GPIO 端口则是由 EXTI 模块中的 CFGL 和 CFGH 寄存器决定。下图列出 EXTI0 位段的值：

EXTI0[2:0]	描述
000	PA0 连接到 EXTI0
001	PB0 连接到 EXTI0
010	PC0 连接到 EXTI0
011	PD0 连接到 EXTI0

因此，在实际应用中应尽量避免同时使用 PA0/PB0…PD0 等相同外部中断线的端口。如果有三个按键需要连接到外部中断线，则建议分别连接到 PA1，PB2，PA3 等这些占用不同 EXTI 口的引脚上，而非 PA0，PB0，PC0，否则还需要在程序中再做甄别。

8.2.4. 注意事项

（1） 由于控制触发中断、触发事件的寄存器是相互独立的，因此同一个外部脉冲可以同时触发以上两种情况。

（2） 由于 Cortex-M0+内核使用了 2 级流水线，需要注意局部的执行时序。在中断服务程序中会存在这方面要求，具体是不要把清中断标志放在中断返回之前的最后一条指令，否则可能会造成中断的二次响应，也可以用一条流水线隔断指令（如“DSB”）来确保清中断标志在中断返回之前执行完成。具体见“例程”说明。

9. 系统配置模块应用指南

9.1. 概述

系统配置模块主要用设置一些系统性的参数，比如 SRAM 锁定、DEBUG 环境等。

9.2. 编程设置

9.2.1. 掉电检测（BOD）

BODMD 位段可以设置检测 VDD 的模式，当 BODMD=10b 时，可以检测 VDD 的上升和下降，当 VDD 上升时，BODF 位为 0，当 VDD 下降时，BODF 为 1。

9.2.2. 低电压复位（LVR）

VLVR[1:0]可以选择三种 LVR 电压，此 LVR 相关寄存器复位后值不变。

9.2.3. NMI 中断开关

NMI 中断属于不可屏蔽中断，优先级在整个芯片中是最高的，而通过使能 SAFR 寄存器的相应位段可以将 CSM、EXTI0、BOD 这三个中断连接到 NMI 中断。

一旦将上述三个中断连接到 NMI 中断后，发生中断时会直接进入到 NMI 中断服务函数，虽然原中断也可以被触发，但由于优先级低于 NMI 中断，因此原中断的服务函数并不会得到执行，这一点在应用程序设计过程中要特别注意。

9.2.4. 晶振引脚作为普通 GPIO

晶振引脚 XTAL1/XTAL2 默认输入输出全关状态，如果要使用外部晶振，需要设置 SAFR 寄存器的 OSCCFG 位，并且设置后直到下一次复位前不能被修改。

OSCCFG[1:0]	描述
00	XTAL1 和 XTAL2 输入输出全关（默认）
01	XTAL1 和 XTAL2 作为外部振荡器接口(晶振和陶振)
10	XTAL1 作为外时钟源输入，XTAL2 作为 GPIO
11	XTAL1 和 XTAL2 输入输出全关（默认）

9.2.5. 仿真引脚作为普通 GPIO

在应用中需要使用大量 IO 引脚时，可以通过设置 SAFR 的 SWJCFG 位段将仿真引脚作为普通 GPIO 使用。

当 SWJCFG 设置为非 101b 时，SW-DP 关闭，但 SWDIO 和 SWCLK 存在特殊情况。即当芯片在正常运行模式下程序将 SWJCFG 设置为非 101b，则此时所有的仿真引脚都被当作 GPIO 引脚，直到下一次 RESET 前不能被修改，也无法进入到仿真模式。当芯片在仿真模式下单步运行时，即使运行到设置 SWJCFG 不等于 101b 这条语句时， SWDIO 和 SWCLK 还是作为仿真引脚，仿真模式也不会被强制退出。

9.2.6. 外设在 debug 模式下运行

当芯片处于仿真模式下时，正常情况外设都会处于停止状态，比如 TIM 的计数器不会继续计数等，通过设置 DBGCR 寄存器的相应位可以使得外设能够继续运行，比如将该寄存器的 BIT9 置 1 则 TIM 就可以正常运行。

DBGCR 寄存器的高 16 位是解锁位，必须为 0x5AA5，设置时需要 32 位数据同时写入到寄存器才能生效。

【注意】在 Debug 时，禁止将仿真口配置为开漏功能，否则会影响仿真调试。在实际项目应用中，如果仿真口，被复用为 TWI，建议在仿真模式下关闭 TWI 的初始化（IO、TWI 模块）。

9.2.7. 低功耗模式

低功耗模式有两种，分别是睡眠模式和停止模式，区别在于后者不但将内核停止，还停止了各外设的时钟，相当于深度睡眠。

在软件处理上，停止模式需要将内核的系统控制寄存器（0XE000ED10）的位 SLEEPDEEP 置 1，然后再执行 WFE 或 WFI 指令。

配置为睡眠模式示例如下：

关闭睡眠模式下不需要运行的外设

配置唤醒源，比如外部中断或其他可以睡眠模式下运行的中断

使用指令 WFI 或 WFE 进入睡眠模式（如果使用 WFE 且之前有中断建议调用两次）

等待 MCU 被唤醒

配置为停止模式示例如下：

配置唤醒源，如外部中断等（停止模式下大部分外设的中断是无法唤醒 MCU 的）

关闭不使用外设的时钟源，模拟模块还需要将模块的使能位也关闭

不使用的 GPIO 引脚可以配置为输入输出全关或模拟输入以节省功耗

设置系统控制寄存器的位 SLEEPDEEP 为 1

调用指令 WFI 或 WFE 进入停止模式（如果使用 WFE 且之前有中断产生建议调用两次）

等待 MCU 被唤醒

以上两种方式中停止模式是功耗最低的，进入此模式后所有的外设时钟都是关闭的，所以普通的外设中断是无法唤醒 MCU 的，比如使用 APB0 时钟的 TIM，由于时钟被关闭，计数器一直无法溢出也就不能产生中断，因此不能用来做中断源。

WFE 和 WFI 是内核的两种进入低功耗的方式，WFE 为事件唤醒，WFI 为中断唤醒，通常能唤醒 WFI 的中断都可以唤醒 WFE，反之不然。

【注意】当系统从 PD 唤醒后，系统时钟跑的是内部 128K，需要手动将系统时钟切换回进 PD 前的时钟。

10. PCA 用户指南

10.1. 概述

可编程计数器阵列 PCA 提供增强的定时器功能，能够提供输入捕捉、比较匹配(输出)、频率可调

方波输出、PWM 调制输出四种增强功能。SH30F9/SB0 系列芯片内置 4 个 PCA 模块（PCA0/1/2/3），每个 PCA 模块均提供 2 路输入输出通道。

SH30F9/SB0 系列芯片其 PCA 特性为：

■16位定时器

■2路独立的输入输出通道

■支持输入捕捉、比较匹配(输出)

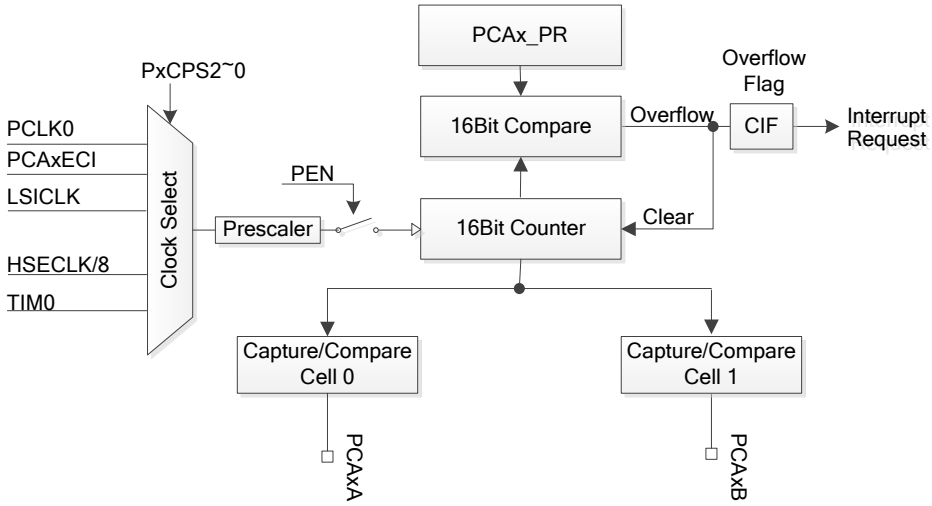
■支持频率可调的方波输出

■16/8位PWM，4种输出模式

■PCA0/1、PCA2/3两两能够级联成32位PCA模块

可编程计数器阵列 PCA 由一个基本计数单元、2 个 16 位捕捉/比较模块组成，每个捕捉/比较模块有独立的引脚（PCAxA/PCAxB），其引脚可以作为捕捉信号输入或波形输出。

注：小下标 x 为 PCA 序号，例如 PCAx（x=0, 1, 2, 3），下文将统一使用 PCAx，不再说明 x 的值。



PCA 原理框图

PCAx 由一个 16 位的周期寄存器 PCAx_PR、一个 16 位的计数寄存器 PCAx_CNT 和一个 16 位的预分频寄存器 PCAx_PSC 组成基本的计数/定时单元。

16 位的计数器有 2 种计数模式，分别为向上计数和中央对齐计数。当寄存器 PCAx_CFGR 中的 SDEN 位为 0 时，计数器工作在向上计数模式，当 SDEN 为 1 时，计数器工作在中央对齐计数模式。

在向上计数模式中，计数器从 0 向上计数到周期寄存器 PCAx_PR 的值，然后重新从 0 开始计数并且产生一个计数器溢出事件，同时状态寄存器 PCAx_SR 中的 CIF 位会被硬件置 1。如果寄存器 PCAx_CFGR 中的 CIE 位被软件设置为 1，则会产生一个中断。

在中央对齐计数模式中，计数器从 0 开始向上计数到周期寄存器 PCAx_PR 值，然后再向下计数到 1 完成一个周期，下一周期再次从 0 开始计数，一直循环。当计数器值为周期寄存器 PR 值时，会产生周期匹配事件，同时状态寄存器 PCAx_SR 中的 PIF 位会被硬件置 1，如果 CFGR 寄存器中的 PIE 为 1，会触发一个中断。当计数器的值计到 0 时会产生一个计数器溢出事件，同时状态寄存器 PCAx_SR 中的 CIF 位会被硬件置 1。如果寄存器 CFGR 中的 CIE 位被软件设置为 1，会触发一个中断。

注：从生成波形的形状上，向上计数模式也可称作锯齿波模式，中央对齐计数模式也可称作三角

波模式。

10.2. 编程设置

10.2.1. IO 设置

在用到 PCA 的相关输入输出功能时，必须要先设置 IO 复用(AF)功能，即要把对应 IO 的复用寄存器（GPIOx->AFRH/L）设置为 PCA 相关的 AF 功能。

10.2.2. 时钟设置

计数器的时钟源可以通过设置寄存器 PCAx_CFGR 中的 CPS[2:0]来选择，分别为总线时钟、PCAxECI 引脚时钟输入、内建的 128KHz RC（LSICLK）、外部高频振荡器/8（HSECLK/8）和 TIM0 溢出，如下表所示。

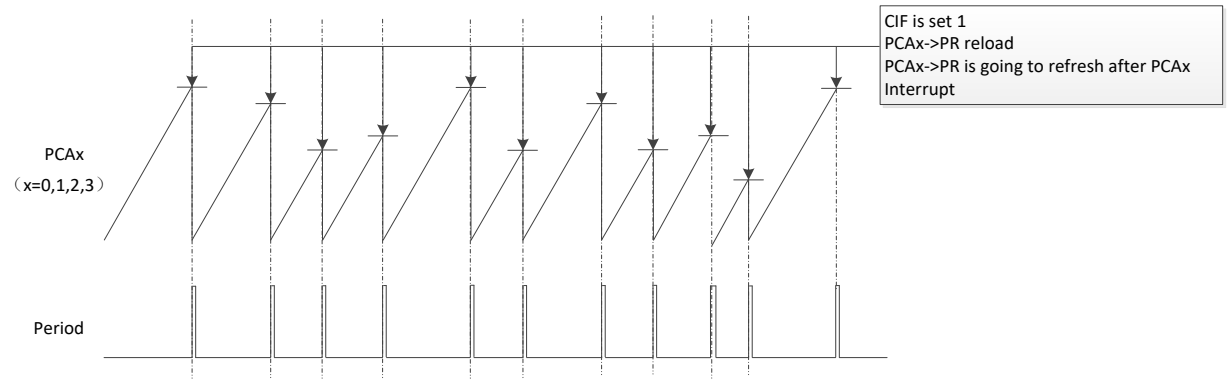
CPS[2]	CPS[1]	CPS[0]	
0	0	0	总线时钟（PCLK0）
0	0	1	外部引脚输入 PCAxECI 时钟下降沿
0	1	0	LSICLK (RC128K)
0	1	1	HSECLK/8 (Crystal/Ceramic 8 分频)
1	0	0	TIM0 溢出
1	0	1	保留项
1	1	0	保留项
1	1	1	保留项

当 CPS[2:0]＝001 时，选择外部 ECI 引脚输入作为时钟源，可以通过设置 CFGR 寄存器中的 ECF[1:0]位段来选择输入时钟源的滤波参数。滤波参数分别为无滤波，8 倍总线时钟，16 倍总线时钟，32 倍总线时钟。

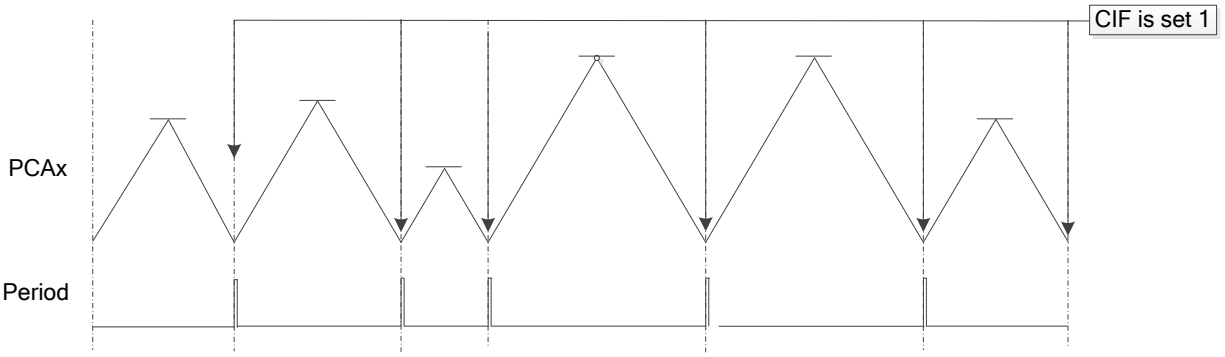
注意：总线时钟周期不得低于计数时钟周期的 4 倍频（计数时钟为总线时钟时除外），否则 PCAx 计数器将不能正确计数。

10.2.3. PCA 计数方式设置

PCAx 的计数器/定时器有一个计数方式可选择的控制位，通过 PCAx->CFGR 寄存器中的 SDEN 为 0 表示锯齿波模式，SDEN 为 1 表示三角波模式。锯齿波和三角波的时序波形图如下所示：



PCAx 计数器在向上计数（锯齿波）模式下的波形图



PCAx 计数器在中央对齐计数（三角波）模式下的波形图

10.2.4. PCA 模式设置

PCAx 计数器挂载 2 路捕捉/比较模块均可以实现增强功能。每个捕捉/比较模块都可被配置为独立工作（注：1 个捕捉/比较模块对应 1 路通道），每个模块在系统控制器中都有属于自己的控制寄存器，这些寄存器用于配置模块的工作方式和与模块交换数据。可以通过配置各自模块 PCAx->CCMR 寄存器中 SM[1:0]两位使能该模块工作在以下 4 种工作模式之一：输入捕捉、比较匹配输出、频率输出、PWM 输出模式。详细见规格书 PCA 章节描述。

10.2.5. 注意事项

1. ***改变 PCAx->PR 值必须保证新的 PCAx->PR 值不小于所有比较寄存器的数值。***
2. ***比较/捕捉模块作为 PWM 输出功能时，若 PCAx->CCR 等于 0x0000，输出一直保持为低电平；若 PCAx->CCR 等于 PCAx->PR，输出则保持为高电平。引脚取反则输出正好相反。***
3. ***PCAx 的所有比较/捕捉模块只能工作于同一计数模式（例：PCA0 的比较/捕捉模块 0 和比较/捕捉模块 1 只能工作在同一计数模式）。***
4. ***级联之后，PCA1 的 PR 值和 CCR0/CCR1 被自动映像到 PCA0 对应寄存器的高 16 位，所以读取只需要读取 PCA0 的 32 位寄存器，其余 PCA 级联同理。***

11. PWM 模块用户指南

11.1. 概述

SH30F9/SB0 系列芯片集成了四个 16 位 PWM 模块。PWM 模块可以产生周期和占空比分别可调整的脉宽调制波形。

PWM 定时器也为 PWMx（x = 0,1,2,3）提供 4 个中断源，在每个 PWM 周期都会产生中断。这样用户可以实现每个 PWM 周期中更改下一次循环的周期或占空比。

有如下特性：

■4×2路带死区控制的互补输出

■提供每个PWMx（x = 0,1,2,3）周期溢出中断

■两路输出极性可独立选择

■提供出错侦测功能可紧急关闭PWM输出

■提供保护寄存器可使重要寄存器免受干扰出错

11.2. 编程设置

11.2.1. IO 设置

在用到 PWM 的相关输出功能时，必须要先设置 IO 复用(AF)功能，即要把对应 IO 的复用寄存器（GPIOx-> AFRH/L）设置为相应的 AF，具体复用信息请参考“复用功能映射表”。

11.2.2. 保护设置

PWM 模块设置有保护寄存器，需要设置 PWM 相关寄存器时，需要先对 PWMx->PWMLOCK 写入 0x5AA5 进行解锁，然后再操作其它寄存器，PWMx->PWMLOCK 写入除 0x5AA5 之外的值时，操作其它寄存器无效，这样可以有效保护 PWM 的输出抗干扰。

11.2.3. 模块设置

PWM 编程分为以下几步：

（1）解锁 PWM 模块。

（2）选择 PWM 模块时钟源。

（3）通过写适当的值到 PWM 周期控制寄存器（PWMPR）或 PWM 占空比寄存器（PWMDR）设置 PWM 周期/占空比。

（4）通过设置 PWMx->CR 寄存器的 PWMSx 位选择 PWMx 输出模式(高电平有效或低电平有效)。

（5）设置 PWMx->CR 寄存器的 PWMEN 使能模块输出

（6）如果 PWM 周期或者占空比需要改变，操作流程如同步骤 3 或者步骤 4 说明。修改后的重载计数器的值在下一个周期开始有效。

（7）PWM 模块上锁。

11.2.4. 注意事项

（1） PWMx 输出关闭时（FLT 发生），可以切断 PWM 的输出，进入无效驱动状态（可设为高电平、低电平和高阻态）。

（2）一旦检测到 FLTx 引脚的高/低电平，内部状态会保持，PWMx 输出会关闭。

（3）当 FLTx 输入信号有效期间，FLTS 位无法清除。只有当 FLTx 输入信号消失后，才能清除 FLTS 状态位。

（4）当 PP[15:0] = 0000H，如果 PWMSA=0，不管 PWMx 占空比为多少，PWMxA（x=0,1,2,3）输出低电平。

当 PP[15:0] = 0000H，如果 PWMSA=1，不管 PWMx 占空比为多少，PWMxA（x=0,1,2,3）输出高电平。

（5）当 PD[15:0] = 0000H，如果 PWMSA=0，不管 PWMx 周期为多少，PWMxA（x=0,1,2,3）输出低电平。

当 PD[15:0] = 0000H，如果 PWMSA=1，不管 PWMx 周期为多少，PWMxA（x=0,1,2,3）输出高电平。

当 PP[15:0] ≠ 0000H 且 PP[15:0] ≤ PD[15:0]，如果 PWMSA=0，PWMxA（x=0,1,2,3）输出高电平。

当 PP[15:0] ≠ 0000H 且 PP[15:0] ≤ PD[15:0]，如果 PWMSA=1，PWMxA（x=0,1,2,3）输出低电平。

12. ADC 模块应用指南

12.1. 概述

SH30F9/SB0 系列芯片内置 1 个逐次逼近方式的 12 位 A/D 转换器， AN4~AN5 两通道转换速率最高可达 1MSPS，其他通道转换速率最高为 100KSPS。基准电压默认使用 AVDD，共有 30 个 ADC 模拟输入通道，支持序列转换方式、间断转换方式和连续转换三种转换方式，带比较功能。不仅支持软件触发 AD 转换，同时支持 4 个 PWM 模块（PWM0/1/2/3）、四个 PCA 定时器（PCA0/1/2/3）、一个普通定时器 TIM2 及外部中断（EXTI0/1/2/3）触发 AD 转换。

12.2. 编程设置

12.2.1. ADC 时钟配置

在系统时钟设置完成后，使能 ADC 模块时钟控制位。

12.2.2. ADC 通道 IO 口功能设置

使能 ADC 通道对应 IO 口的模拟复用功能。

12.2.3. ADC 采样率配置

通过寄存器 ADCON2 可以设置 ADC 的时钟以及采样时间。通过 TADC[3:0]@ADCON2 设置 ADC 的时钟。通过寄存器 TS[3:0]@ADCON2、TGAP[2:0]@ADCON2 和 GAPENx@ADGAPON 设置每个通道的采样时间 tSAMP，tSAMP = (TS[3:0]+ TGAP[2:0]* GAPEN+1) * tAD，其中 TGAP[2:0] GAPEN 为序列中时相邻通道之间时间间隔设置位段， GAPEN 为 ADC 通道 y 的 GAP 时间使能控制位。

对于 12BIT 模式 AD 转换每个通道的 AD 转换时间固定为 15 * tAD。因此每个通道的总共采样转换时间 = tSAMP + 15 * tAD。对于 10BIT 模式 AD 转换每个通道的 AD 转换时间固定为 13 * tAD。因此每个通道的总共采样转换时间 = tSAMP + 13 * tAD。

12.2.4. ADC 转换方式配置

12.2.4.1. 单次转换

ADCTU[1:0] = 00，ADC 配置为单次序列转换模式，一次转换一个序列。序列由单个通道或多个通道组成，对序列进行转换即对序列中的通道进行逐个转换。在硬件上，使得多个信号在同一时间点上转换得以实现（2 通道间最短采样间隔 0.5us，可近似看作同时）。

转换的结果储存在对应的结果寄存器中，序列转换完成的同时，把指定的某个结果寄存器的值与 ADDGT 和 ADDLT 的值比较，并以标志指示比较的结果。

单次序列转换模式下， ADSOC 被置起后开始从通道 0 开始转换，每次单通道转换结束后该寄存器自动加 1，随后进行下一通道转换，当单次序列转换通道个数达到设定值(ADMAXCH[2:0])时，本次序列转换结束，ADSOC 位清零，表明本次序列转换完成，同时自动复位 ADPCH 寄存器为 0；同时 ADINTF 中的 ADIF 位将由硬件置起，此时若 ADCON1 中 ADIE 位为 1（注意：ADC 的中断不要被屏蔽），则将触发序列转换完成中断，ADIFC 标志需要软件清除。

12.2.4.2. 间断转换

ADCTU[1:0] = 01，则 ADC 工作在间断转换工作模式下，ADC 直接从由 ADPCH 寄存器指定的通道开始转换，当转换个数达到(ADMAXCH[2:0])设定值时，本次序列转换结束。ADC 采样通道序号最大为 7，因此当转换通道指针寄存器 ADPCH 超过 7 时，会自动归 0。一个序列转换结束后，如果此时将 ADPCH 软件改写为 0，则下次转换恢复从通道 0 开始；若不改写 ADPCH，则下次转换通道从 ADPCH 的值开始，若通道的预设模拟通道未进行过设置，则可能得到非预期结果，因此程序对该寄存器的改写要小心处理（ADC 转换期间（ADSOC =1），ADPCH 无法修改）。

12.2.4.3. 连续转换

ADCTU[1:0] = 1x 选择连续转换方式，转换完一个序列便进行此序列的下一次转换。连续转换的一些寄存器设置可参考“单次转换方式”，和序列转换不同的是该模式将循环转换序列，一个序列转换完成到下个序列开始转换的时间也是由 TGAP[3:0]控制的。每次序列转换完成的同时，把指定的某个结果寄存器的值与 ADDGT 和 ADDLT 的值比较，并以标志指示比较的结果。当软件清除 ADSOC 位时，AD 转换立即停止。

12.3. 注意事项

为了保证 AD 转换在一定转换速率的条件下能转换出精确的 AD 结果，请务必根据实际应用情况，计算好等效输入阻抗是否匹配，否则将无法获得精度误差在 0.1LSB 之内的转换结果。

ADC 转换前，需要将 ADON@ADCON1 打开 10us 后再将 ADSOC@ADCON1 位置 1，因为 ADC 模块开启后需要一段时间预热才能进行转换。

用于做 AD 转换的通道，IO 口功能必须复用为模拟功能，否则无法获得正确的转换结果。

ADC 转换开启后，不能更改通道参数，包括 TGAP、GAPENx、TS、ADMAXCH、TADC、ADPCH、SEQCHx 等这些核心通道参数，对其修改都将视为无效操作。

SEQCHx 中也可设置相同的通道号，比如将 SEQCHx 中的值全设置为 AN3，结果寄存器中储存的将是不同时间段的 AN3 的采样值。

当 ADIE，ADLIE，ADGIE 的都开启时，ADIF，ADLIF，ADGIF 任意一个置位都能引起中断，且共享一个中断向量，通过判位 ADIF，ADLIF，ADGIF 哪个为 1，来确定具体的中断源并定制特定的操作。

AD 转换序列中相邻通道之间的时间间隔可用于增加 ADC 的采样时间。每个通道的 Gap Time 都可以单独设置，所以 Gap Time 亦可当做每个通道的采样时间。所有通道的 Gap Time 只能设置成一个值，但是每个通道可单独设置是否使能 Gap Time。第一个通道也可以设置 Gap Time，此时间亦可当做采样时间。

任何硬件触发单次序列、间断转换和连续转换过程中，又来了一次硬件触发，那么后一次硬件触发请求将被忽略。

应用比较功能时，ADDGT 和 ADDLT 写入值立即生效，比较时会采用 ADDGT 和 ADDLT 最近一次的更新值。

13. TIM 基本定时器应用指南

13.1. 概述

SH30F9/SB0 系列芯片内部有 3 个基本的 16 位定时器和 1 个基本的 32 位定时器。每个定时器都能以总线时钟运行，还能以内部低频 RC(128KHz 的 LSI/4)运行。

每个定时器都可以通过实际的 GPIO 引脚输出频率可设置的方波，也可以通过 GPIO 引脚从外部输入时钟源驱动定时器的计数单元。

13.2. 编程设置

13.2.1. 基本定时应用

基本定时器可以配置成简单的定时功能，在实际应用中多用于系统时基或软件定时。

13.2.2. 输出频率可设置的方波

通过设置 GPIO 引脚利用为 TIM 功能，然后使能 CR 寄存器的 TC 位，就可以根据定时器的溢出时间翻转 GPIO 引脚，产生一个方波，需要改变频率只需要修改定时器的溢出时间即可。

13.2.3. 注意事项

1）溢出标志位的产生是在计数器 TCNT 从周期寄存器（TPR）里的值变为 0 的时候产生的，因此实际上一个周期的实际时钟数为 TPR+1，也就是应用中需要周期为 100 时，TPR 应该设置为 99。

2）如果设置计数器 TCNT 的初始值大于周期 TPR 的值，那么使能定时器后，计数器会一直向上运行到 0Xffff(TIM3 会运行到 0xFFFFFFFF)然后溢出为 0，在下一个周期才会在 0~TPR 之间循环运行。

3）计数器寄存器、预分频寄存器及周期寄存器都在定时器停止时修改。

4）Timer 时钟选择外灌时钟时时，需要确保总线时钟的频率最慢为 3 倍的外灌时钟频率。

5）TIM3 的使用方法和 TIM0-2 一致，区别是 TIM3 为 32 位计数，而 TIM0-2 为 16 位计数。

6）Timer 时钟源选择 Tx 口输入时，不要有对 TCNT 进行写的操作。若有清零 TCNT 然后计数的需求，可以用读取前后两次的 TCNT 值做差值来代替。

14. UART 用户指南

14.1. 概述

SH30F9/SB0 系列芯片提供 6 组 UART。支持灵活的 IO 功能映射。

每组 UART 有四种工作方式，自带波特率发生器，支持硬件奇偶校验，UART0~2 可配置 FIFO，支持基于 FIFO 的数据收发。

另外还支持 1/2 位停止位可设，支持 LIN 总线同步间断的硬件产生和检测。增强功能包括帧出错检测及自动地址识别。

14.2. 编程设置

14.2.1. 发送

UART 发送逻辑内部有 TDR 与 TDR shift 两级寄存器，所以客户首次可以连续写入 2byte 发送数据。开启 UART 的时候，发送中断标志位 TI 默认为有效，直到 TDR 和 shift 都填充好数据后才会自动清零，随后中断 TI 将在 TDR 的数据被 load 到 shift 寄存器后变为空时产生。TC 标志位置位发生在 TDR 和 shift 寄存器内数据都被发空时，该标志位会随数据的写入自动清 0。

14.2.2. 接收

UART 接收电路内部有 RDR 与 RDR shift 两级寄存器，用户在接收到数据后需要及时取走，否则当 RDR 内存有数据时，shift 内数据也接收完毕后会被冲掉。所以用户最迟需要在 shift 数据接收完成前取走 RDR 数据。RI 的标志在接收完完整一 byte 数据后产生，RDR 内数据被读空后，RI 会自动清 0。

【注意】UART 单字节数据接收完毕后，务必在下一个单字节数据接收完毕之前取走，否则会出现数据丢失或者接收溢出。

14.2.3. IO 配置

由于 TXDx/RXDx 引脚与 I/O 功能复用。如果 AF 功能选择了 UART，则 RXDx 引脚自动设为 UART 输入，但是内部上拉电阻需要提前手动开启；TXDx 引脚自动设为 UART 输出，但是开漏输出还是推挽输出需要提前初始化好。注意，若与仿真口复用，需要在 SYSCFG 里先关闭仿真口，否则 AF 配置不生效。

14.2.4. LIN 模式

LIN 模式下，检测间断帧与接收数据不可以同时进行，正确使用流程是先检测同步间断，然后在 LBD 中断下切换到接收数据模式，接收完成后再切回同步间断检测模式。

【注意】UART LIN 总线模式同步间隔检测时，帧出错标志置起属于正常情况，无需特别处理，只需清零即可。

14.2.5. 中断

使用中断模式时，建议 TIE 在初始化 UART 后发送数据前开启，否则在使能 TEN 前开启了 TIE，中断服务程序会先搬运好 2byte 数据到 TDR 和 shift，等到 TEN 使能时，发送出去。

14.2.6. 奇偶校验

使用奇偶校验功能时，其优先级最高，多机通讯及自定义第九位功能将不能使用。

14.2.7. 错误标志

发送冲突，接收溢出，奇偶校验错误，帧错误等错误标志位再发生时会置位，但是不会触发中断发生，客户如果要使用，需要在 UART 中断中自行查询处理。

14.2.8. 其他注意

为了避免接收端误判，在 UART 发送初始化程序中，建议将相应 TXD 的 I/O 状态设为输出高电平。

15. SPI 用户指南

15.1. 概述

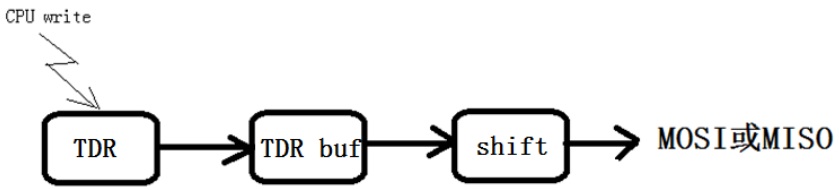
SH30F9/SB0 系列芯片提供 2 组 SPI。支持灵活的 IO 功能映射。

每个 SPI 都支持工作在全双工模式，主/从机模式，可编程主时钟频率（最高支持总线频率 2 分频）串行时钟的极性和相位可编程，每次发送数据字位宽 8、16 和 32bit 可选，大小端模式可选。SPI0 可配置 FIFO，支持基于 FIFO 的数据收发。

15.2. 编程设置

15.2.1. 发送

SPI 的发送电路内含有 TDR、TDR buffer 和 shift 三级寄存器（后两者用户不可操作）；用户首次写入可以写入 2 笔数据。开启 SPI 的时候，发送中断标志位 SPTI 默认是置位的（表示 TDR 为空），直到 TDR 和 TDR buffer 都填充好数据后才会清零。SPTI 的再次置位是在 TDR 中的数据被加载到 TDR buffer 后发生，表示可以再次往 TDR 中写入 1 笔数据。三级寄存器示意图如下：



【注意】第 1 笔数据会写穿到 TDR buf 中。

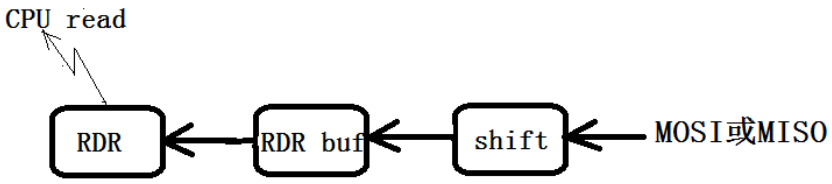
【注意】SPI 没有独立的发送使能开关，在 CPU 写入 1 笔数据后 shift 就开始发送，不存在一种把两级寄存器填满以后才开始发送的情况，流程总是处于边写入边发送的过程中。

【注意】对 TDR 写入即清零 SPTI。

【注意】不允许连续写 TDR，应该每次只写一笔数据。

15.2.2. 接收

SPI 接收电路内部有 RDR、RDR buffer 与 shift 三级寄存器，用户在接收到数据后需要及时取走，否则当 RDR、RDR buffer 内存有数据时，shift 内数据也收满后，后续再来数据会发生接收溢出事件（RXOV），该后续数据会被丢弃。用户最迟需要在 shift 数据收满后且新数据到来前取走 RDR 数据。SPRI 的标志在 RDR 存储 1 笔数据后置位（表示 RDR 为满），RDR 内数据被读空后，SPRI 会自动清 0，若下层 buffer 有数据，该数据会被自动推到 RDR 寄存器，且重新置位 SPRI。



【注意】shift 是发送和接收共用的，而 buffer 和 data register（TDR/RDR）是独立两套的。shift 的共用在 data sheet 中有明确的说明。

【注意】对 RDR 读出即清零 SPRI。

15.2.3. IO 配置

由于 SPI 引脚与 I/O 功能复用。如果 AF 功能选择了 SPI，则信号输入引脚自动设为 SPI 输入，但是内部上拉电阻需要提前手动开启；信号输出引脚自动设为 SPI 输出，但是开漏输出还是推挽输出需要提前初始化好。

15.2.4. 快速从机模式

当 SPI 模块选择 CPHA=0 的从机模式下，$\overline{SS}$引脚必须作为片选信号使用，且每发送 1byte 前拉低，发送完 1byte 后要拉高，如此循环。

【注意】必须使用$\overline{SS}$引脚的原因是在 CPHA=0 时，从设备没有时钟边沿来触发数据加载操作，只能利用$\overline{SS}$信号线来完成数据加载。 CPHA=1 就不存在这一问题，可以设置 SSDIS=1 来屏蔽$\overline{SS}$信号。

为了避免对$\overline{SS}$引脚的控制降低通信效率，引入了从机快速模式（SPSFF），只需首次发送时把$\overline{SS}$引脚拉低一下，后续的数据自动由前一笔数据的 SCK 后边沿加载，省略了$\overline{SS}$引脚控制。

【注意】如果是全双工通信，对从机而言，由于不知道什么时候 SCK 会到来，因此需要预先在 TDR 中加载一笔数据，否则会造成从机往主机方向第一个数据的延迟，对这类情况往往建议用 CPHA=1 模式。

15.2.5. 中断

使用中断模式时，SPTIE 需要在初始化 SPI 后，发送数据前开启。在发生 MODF 错误时，需在中断中使用软件切为从机并关闭 SPI。若 SPI 需要切为从机接收数据，则需要重新使能 SPI。若 SPI 还需要作为主机收发数据，则需要先检测 SS 电平为高才可以设置为主机重新开启 SPI 通信。

15.2.6. 错误标志

发送冲突，接收溢出的错误标志位不会触发中断，如有需求需要用户在程序中自行检查处理。

16. TWI 用户指南

16.1. 概述

TWI（Two-wire Serial Interface）接口是对 I2C 总线接口的继承和发展，完全兼容 I2C 总线。

TWI 由一根时钟线和一根数据线组成，以字节为单位进行传输，可兼容 SMBus 总线规范，自动对字节传输进行处理，并对串行通信进行跟踪。

SCL/SDA 是 TWI 总线的信号线。SDA 是双向数据线，SCL 是时钟线。在 TWI 总线上传送数据，首先送最高位(MSB)，由主机发出起始信号，然后由主机发送一个或多个字节的数据，数据传送完毕，由主机发出停止信号，完成一次最简单的 TWI 传输。

16.2. TWI 总线配置

16.2.1. I/O 口配置

TWI 总线要求配置为开漏输出，注意开漏输出时外部上拉限制为 VDD 或略超 VDD。

```
GPIOA ->AFRH.BIT.AFR14 = 4;          // PA14 = AF4 ~ SDA0
GPIOA ->AFRH.BIT.AFR13 = 5;          // PA13 = AF5 ~ SCL0
GPIOA ->OTYPER.BIT.OT14 = 1;          // PA13/14: OD output
GPIOA ->OTYPER.BIT.OT13 = 1;
GPIOA ->PUPDR.BIT.PHDR14 = 1;        // PA13/14: pull high，此处用内部上拉示例
GPIOA ->PUPDR.BIT.PHDR13 = 1;
```

16.2.2. 波特率配置

```
// 预分频，0:64 分频， 1:16 分频， 2:4 分频， 3:1 分频
TWI->CR.BIT.CR = 3;
// 波特率公式：Ftwi/(16+2*CR*BRT)，设置波特率 100Kbps 时，Ftwi=24MHz，要求 CR*BRT=112，取 CR=1，BRT=112
TWI->BRT = 112;
```

16.2.3. 通信地址配置

```
// TWIAMR=0，不开启屏蔽。ADDR[6:0]=0111011b，本机地址。GC=0b，禁止响应通用地址。
TWI->ADDR.V32 = 0x0076;
```

16.3. 四种基本通信模式

SH30F9/SB0 系列芯片的 TWI 通讯是以底层驱动电路和以中断为基础的应用软件共同完成的。诸如接收到一个字节或发送一个起始条件的所有总线事件均会产生一个中断。所以在字节传输期间，应用软件可以进行其它的操作。

TWI 的四种基本操作模式：

- 主机发送模式：本机作为主机，发送数据
- 主机接收模式：本机作为主机，接收数据
- 从机发送模式：本机作为从机，发送数据
- 从机接收模式：本机作为从机，接收数据

所有的操作都是从这四种基本操作模式组合而成的。

【注意】TWINT 置位期间 TWI 总线不进行收发操作，状态也不会更新，清除标志后状态才能更新。

【注意】TWINT 置位只和本机状态变化有关，可能是本机完成了某个发送，也可能本机完成了某个接收，都会导致状态变化。

【注意】在初始化阶段，AA=1 表示设置为从机模式，在通信过程中，AA=1 表示回复 ACK 信号。

【注意】TWI 总线中，起始条件和停止条件都由主机发出。

因为 TWI 是收发合用一条双向数据线 SDA，因此在通信时要严格按照时序进行，在 data sheet 中有 4 种基本通信模式的流程示意图，所有的 TWI 协议都是从这 4 种基本模式衍生出来。

16.4. TWI 中断和超时

SH30F9/SB0 系列芯片的 TWI 有 3 个中断源：通讯中断、总线超时、SCL 高电平超时。对应的三个中断标志分别是 TWINT、TOUT、TFREE。

SH30F9/SB0 系列芯片的总线超时（TOUT）功能和 SCL 高电平超时（TFREE）功能是默认开的，无法关闭。

TOUT 是 SCL 低电平超过一定时间后发生，表示某个主机或从机能够把总线时钟拉低的时间上限，超过此时间将产生总线超时事件，硬件自动释放总线，TOUT 标志位置起。

【注意】TOUT 超时功能是固定开启的，如果长时间拉低 SCL，必然会发生 TOUT 事件。TOUT 的默认超时时间是 25000 个总线时钟。

TFREE 是 SCL 高电平超过一定时间后发生，表示总线处于“空闲”状态，发生 TFREE 事件后，硬件自动释放总线，TFREE 标志位置起。

【注意】TFREE 超时功能是固定开启的，如果长时间拉高 SCL（因为 SCL 默认上拉，此状态表示 SCL 空闲），必然会发生 TFREE 事件。TFREE 的默认超时时间是 HOC×1024 个总线时钟（HOC 默认值 255）。

【注意】ETOT 和 EFREE 是控制 TOUT 和 TFREE 事件发生时是否产生中断的使能位，不是模块功能使能位（再次强调：模块功能是固定开启的）。如果 ETOT 和 EFREE 关闭，即使超时标志置起也不会产生中断。

【注意】TWINT 是通讯中断标志，TWI 的状态改变时会引起置位，但从其他状态进入 F8 状态不会引起置位。

17. 附录

17.1. 附录 1：KEIL MDK 中函数定位和调试方法

以 SH30F9010 为例进行说明：

Code Area: 0x0000 0000~0x0003 FFFF

SRAM:0x2000 0000~ 0x20003FFF

17.1.1. 在 RAM 中调式整个程序

要求: 所调试程序必须小于 RAM 的大小。

目的: 将 SRAM 分成 2 个部分，将整个程序放到 SRAM 前半部分运行，后半部分存放变量。

步骤：

1. 在项目目录下增加 Debug 初始化文件并保存为”run_in_ram.ini”。文件为纯 ASCII 文本格式，内容为

```
FUNC void Setup (void) {
     SP = _RDWORD(0x20000000);
     PC = _RDWORD(0x20000004);
  _WDWORD(0xE000ED08, 0x20000000);
}
FUNC void OnResetExec (void)  {
     Setup();
}
load %L incremental
Setup();
g, main
```

语句说明：

SP=_RDWORD(0x20000000): 从 0x20000000 中读取 1 个字送给 SP

PC=_RDWORD(0x20000004): 从 0x20000004 中读取 1 个字送给 PC

_WDWORD(0xE000ED08,0x20000000): 设置中断向量表起始地址

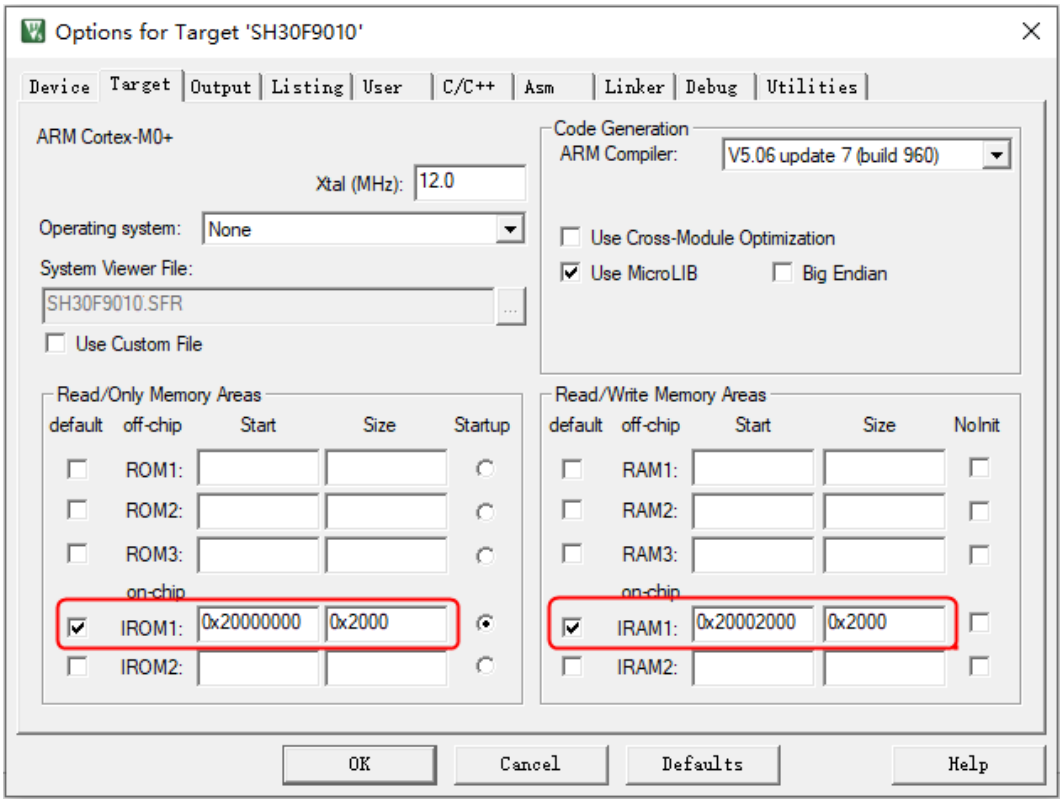
OnResetExec(); 此函数在按 RESET 按钮时调用

Load %L incremental : 添加文件的调试信息，%L 表示 Link 的 Output 文件

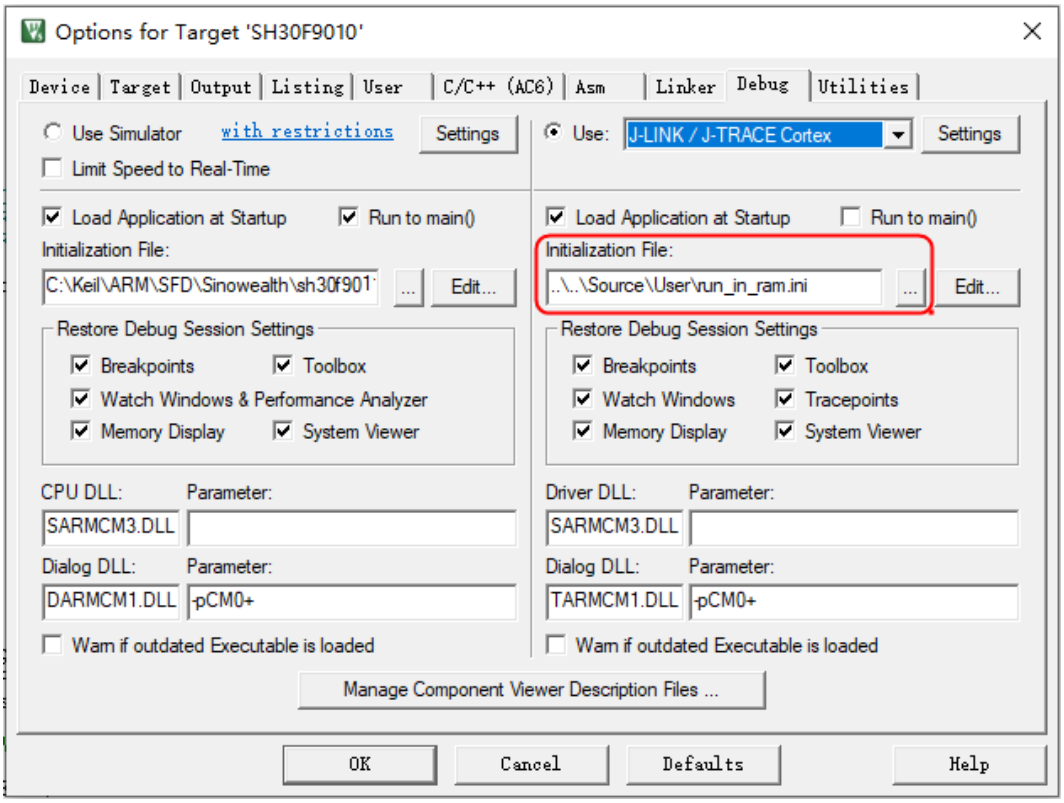
g, main 表示执行到 main 函数

**函数外的语句在进入 debug 时调用。

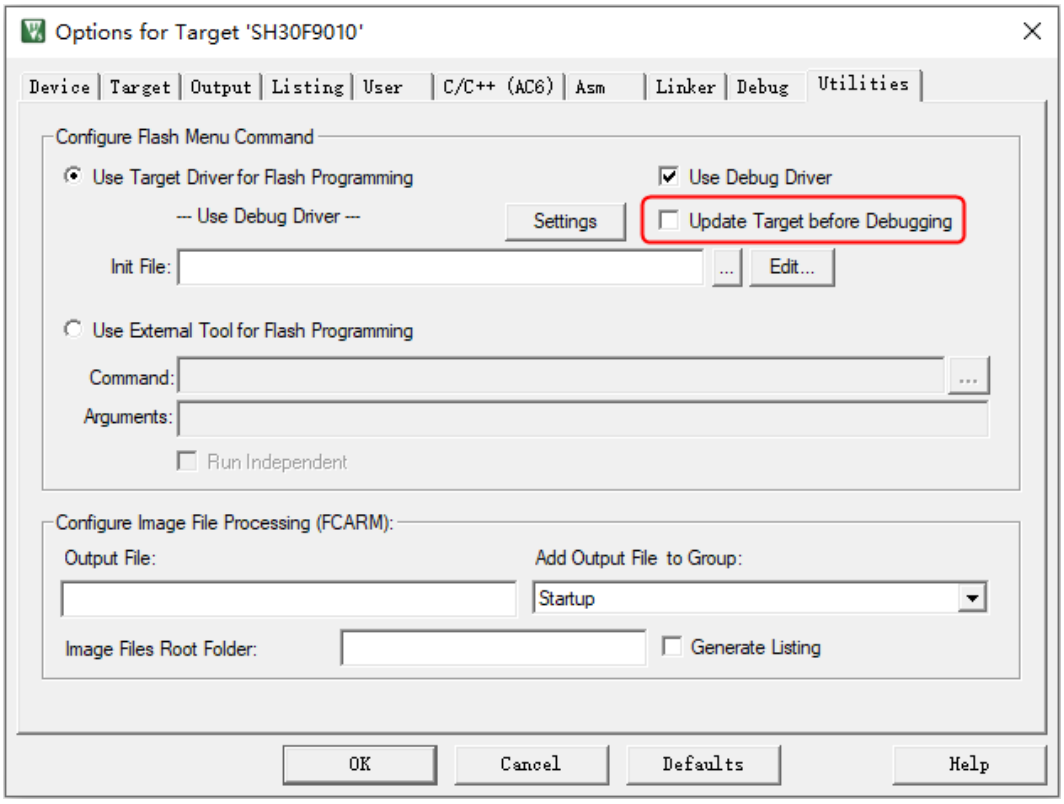
2. 修改 IROM 设置及调整 RAM 设置如下



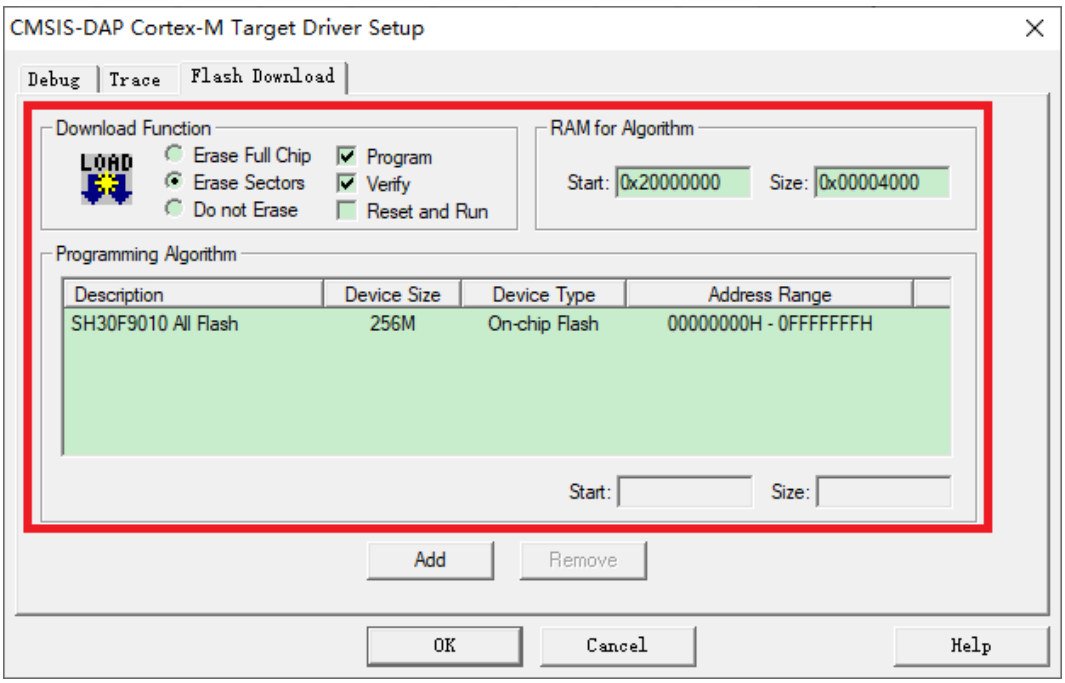
3. 修改 DEBUG 选项如下：在 Initialization File 中加入之前保存的 run_in_ram.ini



4. 修改 Utilities 选项如下：取消 Update Target before Debugging



5．修改 Flash Download 设置如下



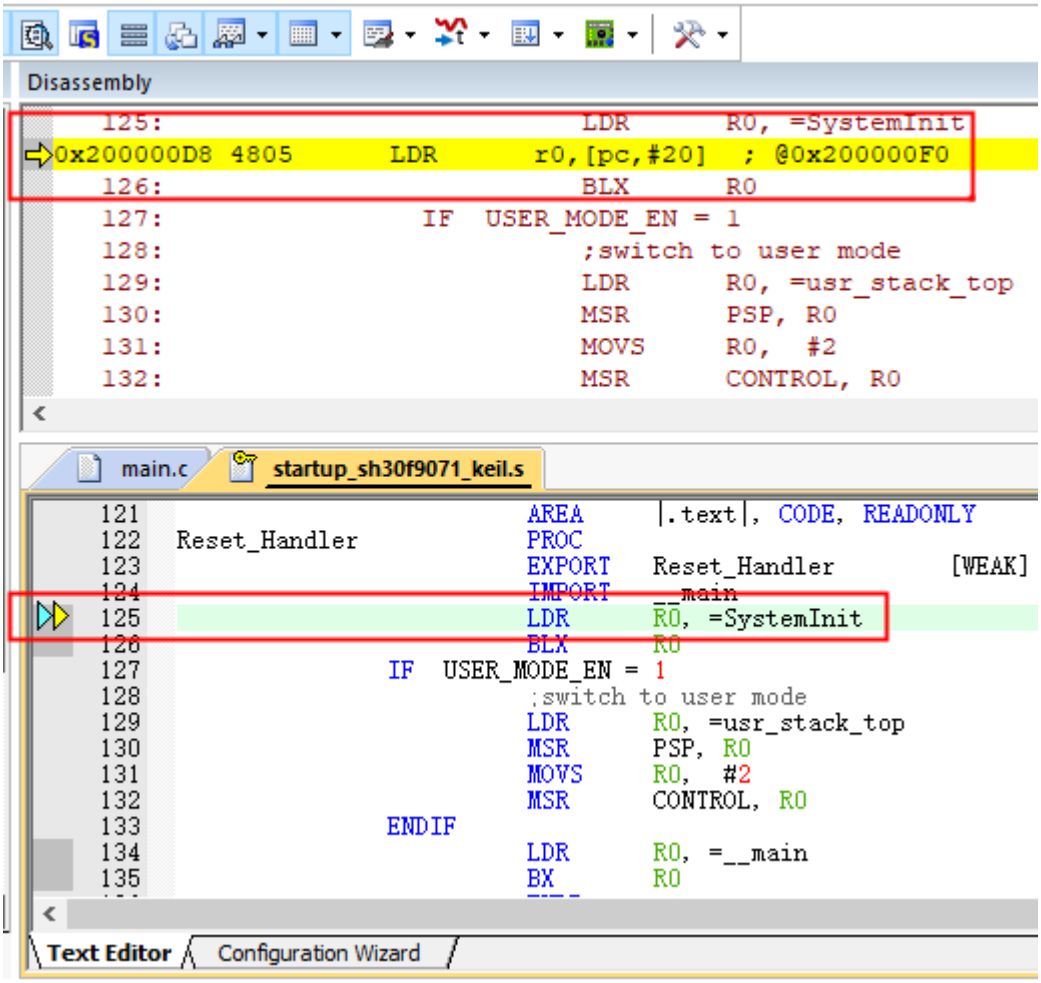
6．Rebuild 项目后直接按进入 Debug 按钮。会出警告“没有 Flash 操作”， 点击确认

7．已经正常运行在 RAM 中了。如下图

注意事项:

不能用 RESET 按钮进行复位，必须退出 Debug 后重新进入。因为复位后会从 Code 区 0 的位置取 SP， 4 的位置取 PC。

程序必须将中断向量表的位置重定位到 0X20000000，否则中断或异常无法正确定位。



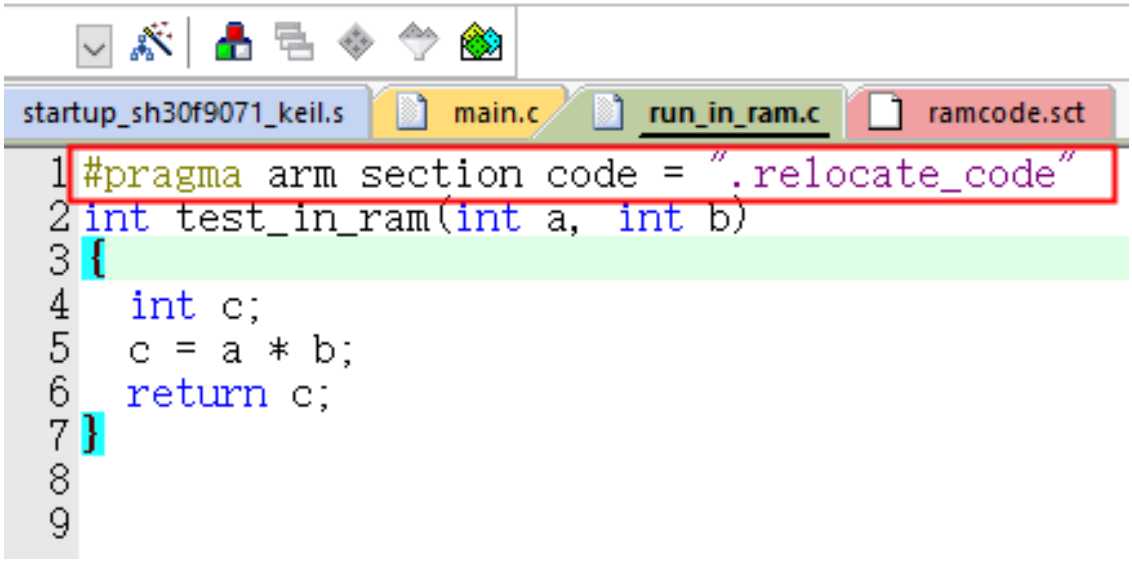
17.1.2. 将关键程序放到 RAM 中运行

要求: 需要运行在 RAM 的程序必须小于 RAM 的大小。

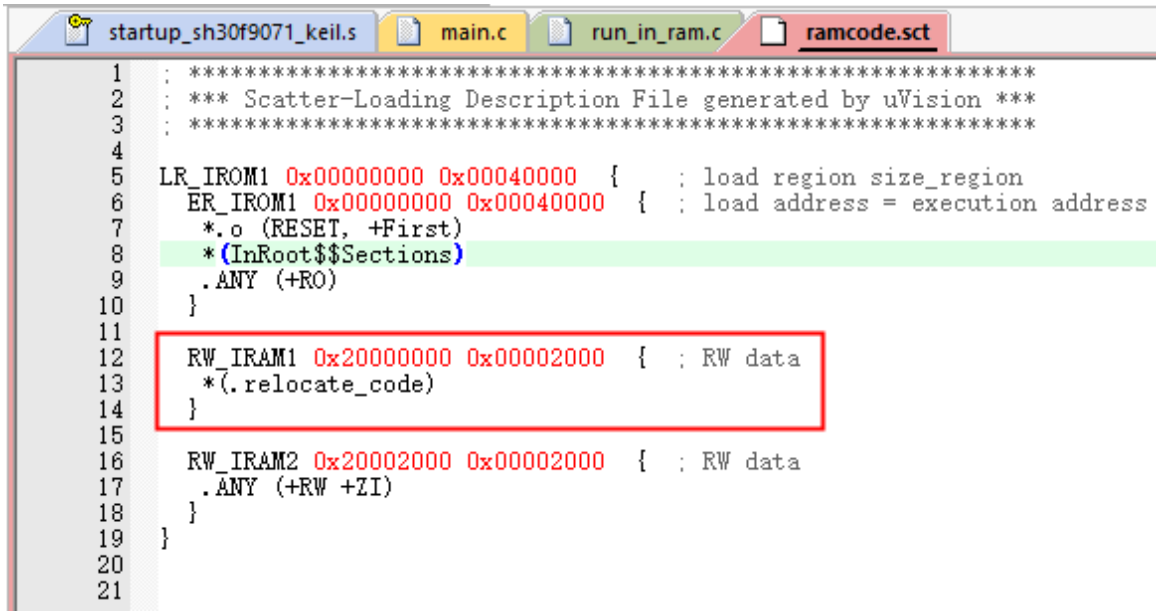
目的: 提高关键代码运行速度

步骤：

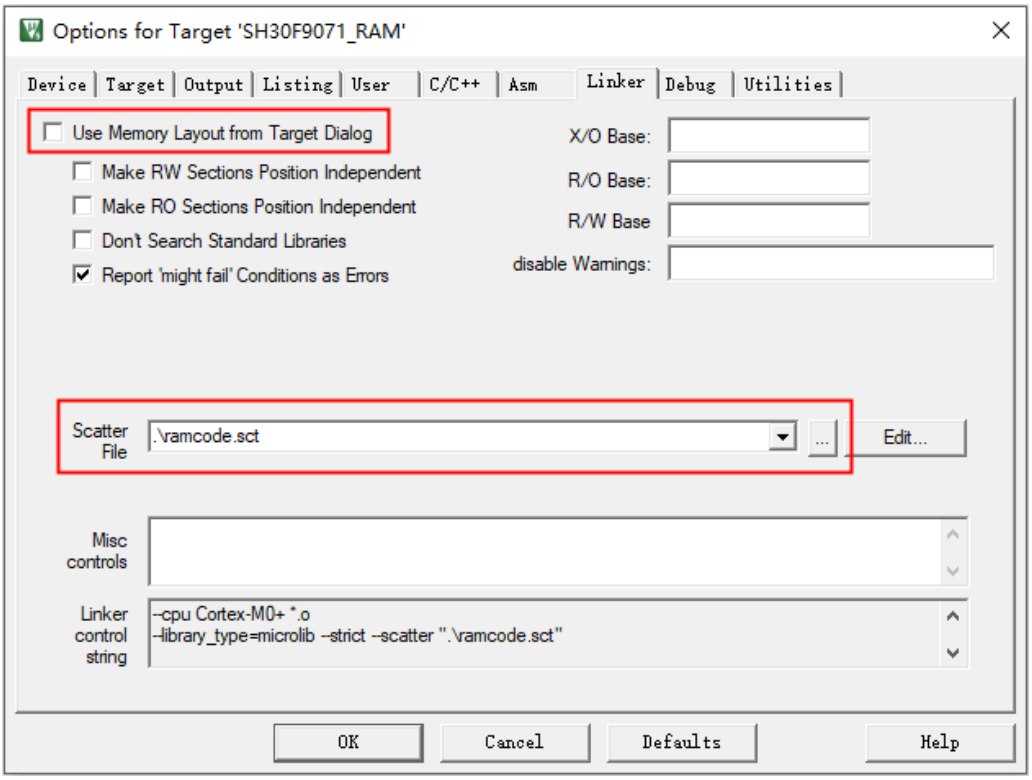
1．将需要放到 RAM 中运行的函数放到一个文件中，在最前面加上定义 section 的编译器指令。（section 名字可以自己定义）如图：



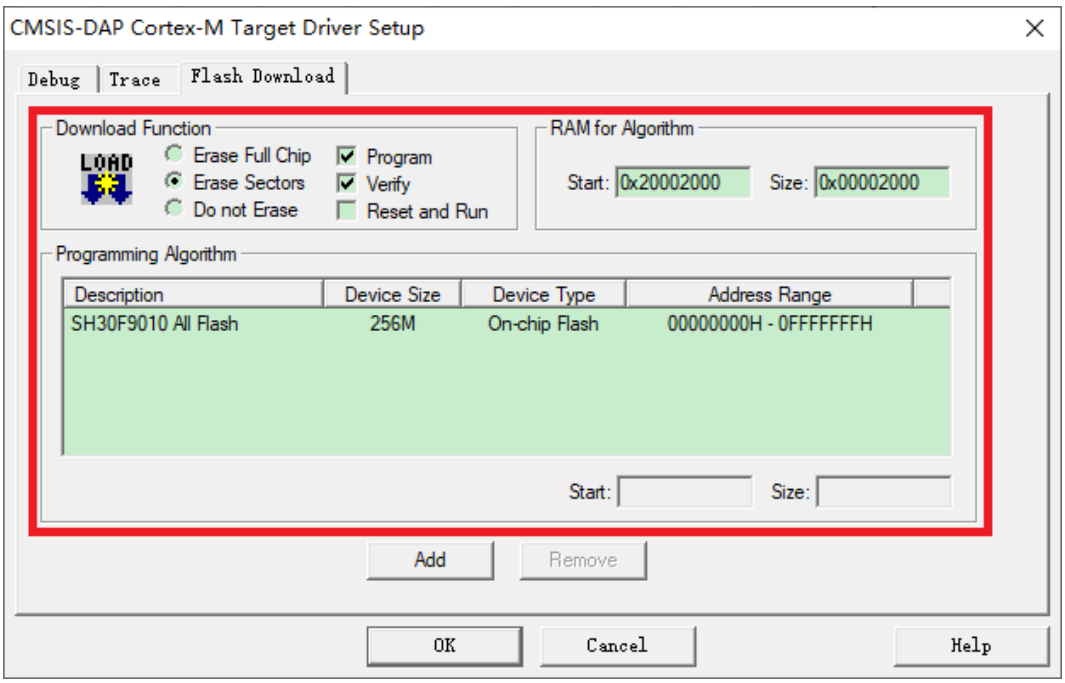
2. 增加 Link 控制文件 ramcode.sct。如图



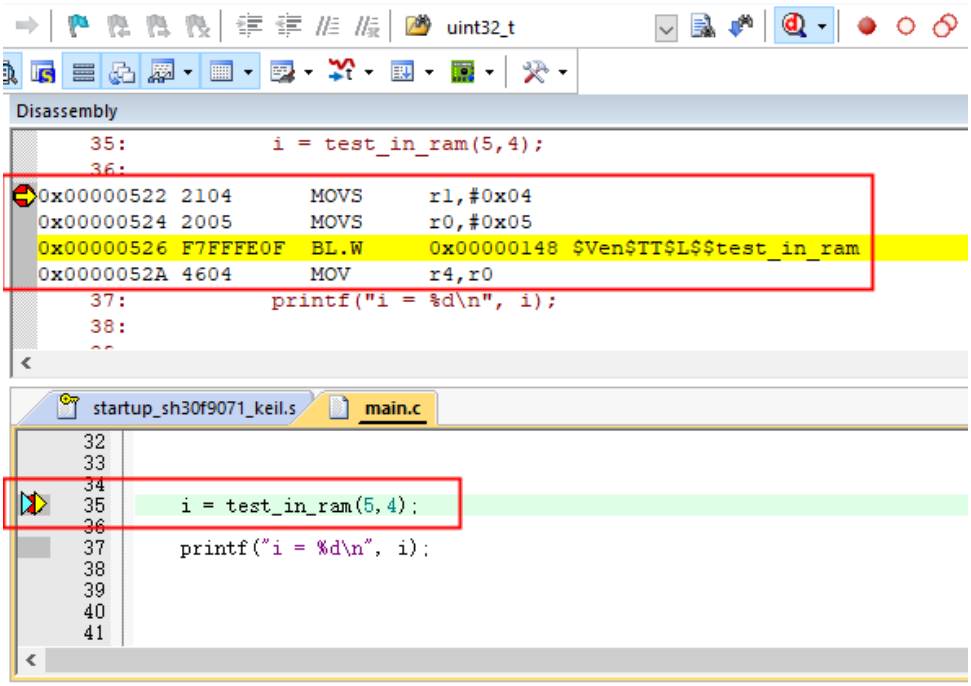
3. 设置 Link 选项，如图



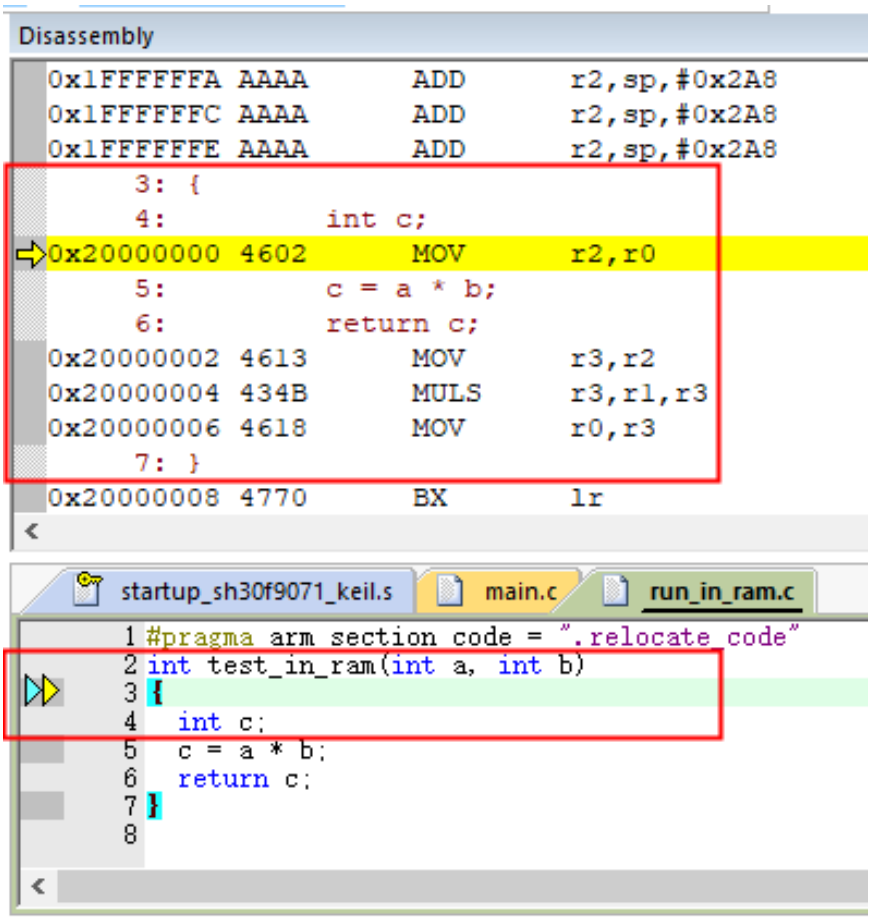
4. 修改 Flash Download 设置如下



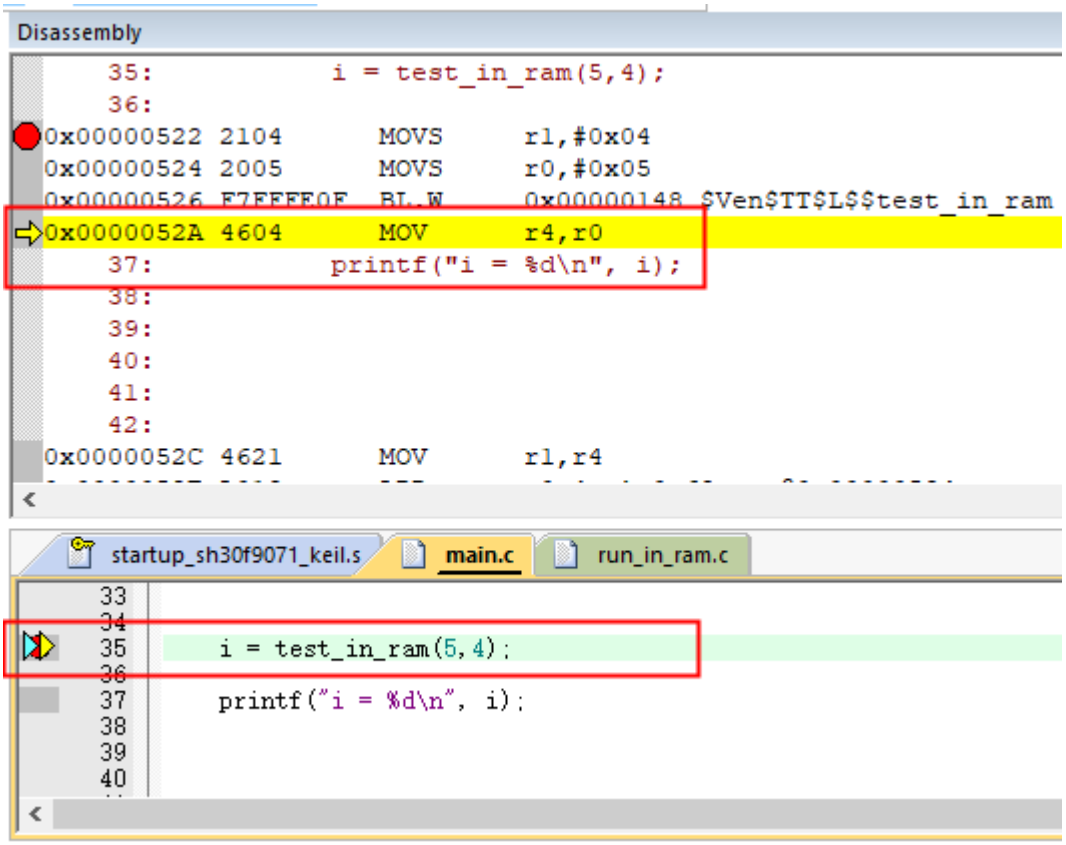
5. Rebuild 项目后按 Download 按钮下载程序，然后按进入 Debug 按钮进入 Debug 界面。运行到调用该函数的地方可以看到这时是运行在 Flash 中



6. 运行 Step Into 后看到，已经跳转到 RAM1 中。



7. 再运行 Step Out，可以看到又跳回 Flash 区域



17.1.3. 将某个函数固定到某个位置

1. 给函数制订一个 section 名字

```
5
6  void OutputInfo1(void) __attribute__ ((section("XNTEST1")));
7  void OutputInfo2(void) __attribute__ ((section("XNTEST2")));
8
9  void OutputInfo1(void)
10 {
11     printf("XN test 1,you can see this infomation.\r\n");
12 }
13
14 void OutputInfo2(void)
15 {
16     printf("ERROR:XN test 2,if you see this infomation,code is error.\r\n");
17 }
18
```

也可以用#pragma arm section code="XNTEST1"为整个文件中的所有函数定一个同名的 section。

2. 在 scatter 文件中指定地址

```
1
2  LR_IROM1 0x00000000 0x00040000  {     ; load region size_region
3    ER_IROM1 0x00000000 0x20000 {   ; load address = execution address
4     *.o (RESET, +First)
5     *(InRoot$$Sections)
6     .ANY (+RO)
7    }
8    ER_IROM2 0x020000 FIXED{
9     *.o (XNTEST1)
10   }
11   ER_IROM3 0x030000 FIXED{
12    *.o (XNTEST2)
13   }
14   RW_IRAM1 0x20000000 0x00002000  {   ; RW data
15    .ANY (+RW +ZI)
16   }
17 }
18
```

如图所示函数 OutputInfo1 被固定在 0x20000 的位置

如图所示函数 OutputInfo2 被固定在 0x30000 的位置

17.2. 附录 2：将同类数据自动导出到一张表中

17.2.1. 定义一个输出宏

例如:

```
#define SECTION(x)                __attribute__((section(x)))
#define RT_USED                   __attribute__((used))
typedef int (*drv_func) (void* param);
struct func_item{
   drv_func     func;
   const char*   desc;
};
#define EXPORT_FUNC_TABLE(name,desc)  const char _drv_##name##_desc[] = #desc; \
                                 const struct func_item _drv_##name##_cmd SECTION("MySymTab") = \
                                  {  \
                                      (drv_func)&name, \
                                      _drv_##name##_desc \
                                  };
```

17.2.2. 导出调用函数

在需要导出的地方调用此宏

例如:

```
static int func1 (void* param)
{
   return 3;
```

```
}
EXPORT_FUNC_TABLE(func1, func desp1);


static int func2 (void* param)
{
   return 1;
}
EXPORT_FUNC_TABLE(func2, func desp2);


static int func3 (void* param)
{
   return 0;
}
EXPORT_FUNC_TABLE(func3, func desp3);


static int func4(void* param)
{
   return 5;
}
EXPORT_FUNC_TABLE(func4, func desp3);
```

17.2.3. 查表使用方法

**注：在 KEIL MDK 中会自动添加表头标识 MySymTab$$Base 及表尾标识 MySymTab$$Limit

```
void my_func(void)
{
   struct func_item *ps;
   struct func_item *pe;
   extern const int MySymTab$$Base;
   extern const int MySymTab$$Limit;
   ps = (struct func_item*) MySymTab$$Base;
   pe = (struct func_item*) MySymTab$$Limit;
   while(ps != pe)
   {
      ps->func(0);
      ps++;
   }
}
```

17.3. 附录 3：将变量固定到 RAM 某个位置

17.3.1. 直接在变量定义处指定地址

```
uint32_t g_var1 __attribute__((at(0x20000000))) = 0x33445566;
uint32_t g_var2 __attribute__((at(0x20000004)));
```

17.3.2. 直接在变量定义处指定所属 section

在变量定义处指定所属 section，然后用 linker 的 scatter file 将此 section 固定到某个位置:

```
uint32_t g_fvar1  __attribute__((section(".fixed1"))) = 0x12345678; /* RW */
uint32_t g_fvar2  __attribute__((section(".fixed1"))) = 0x87654321; /* RW */
uint32_t g_fvar3  __attribute__((section(".fixed1"))) = 0x87654321; /* RW */
uint32_t g_fixed1 __attribute__ ((section(".fixed_var")));
```

定义变量 g_fvar1, g_fvar2,g_fvar3 属于 section “.fixed1”，变量 g_fixed1 属于 section “.fixed_var”。

然后在 scatter file 中定义如下:

```
RAM_IRAM_1 0X20000400 0x0000400{
  *(.fixed_var);
 }
 RAM_IRAM_2 0X20000800 0x0000400{
  *(.fixed1);
 }
```

Section “.fixed_var”中的变量就被定义到了 0x20000400 起始的位置

Section “. fixed1”中的变量就被定义到了 0x20000800 起始的位置

17.3.3. 将文件中所有变量指定到固定位置

在源文件头定义 section，属性为 rwdata。

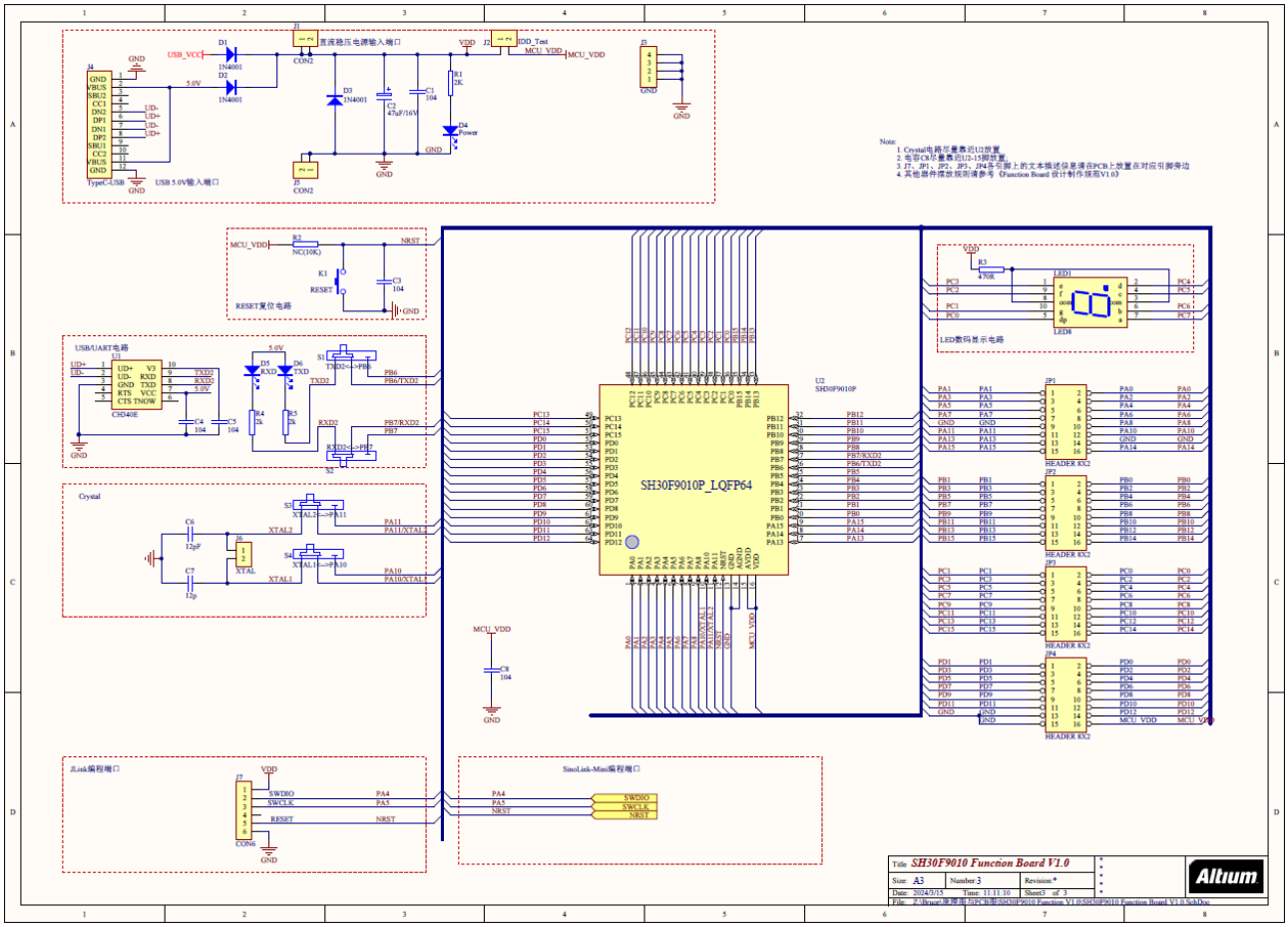
```
#pragma arm section rwdata=".fixed_var"
#pragma arm section zidata=".fixed_var"
uint32_t g_fixed1;
uint32_t g_fixed2;
```

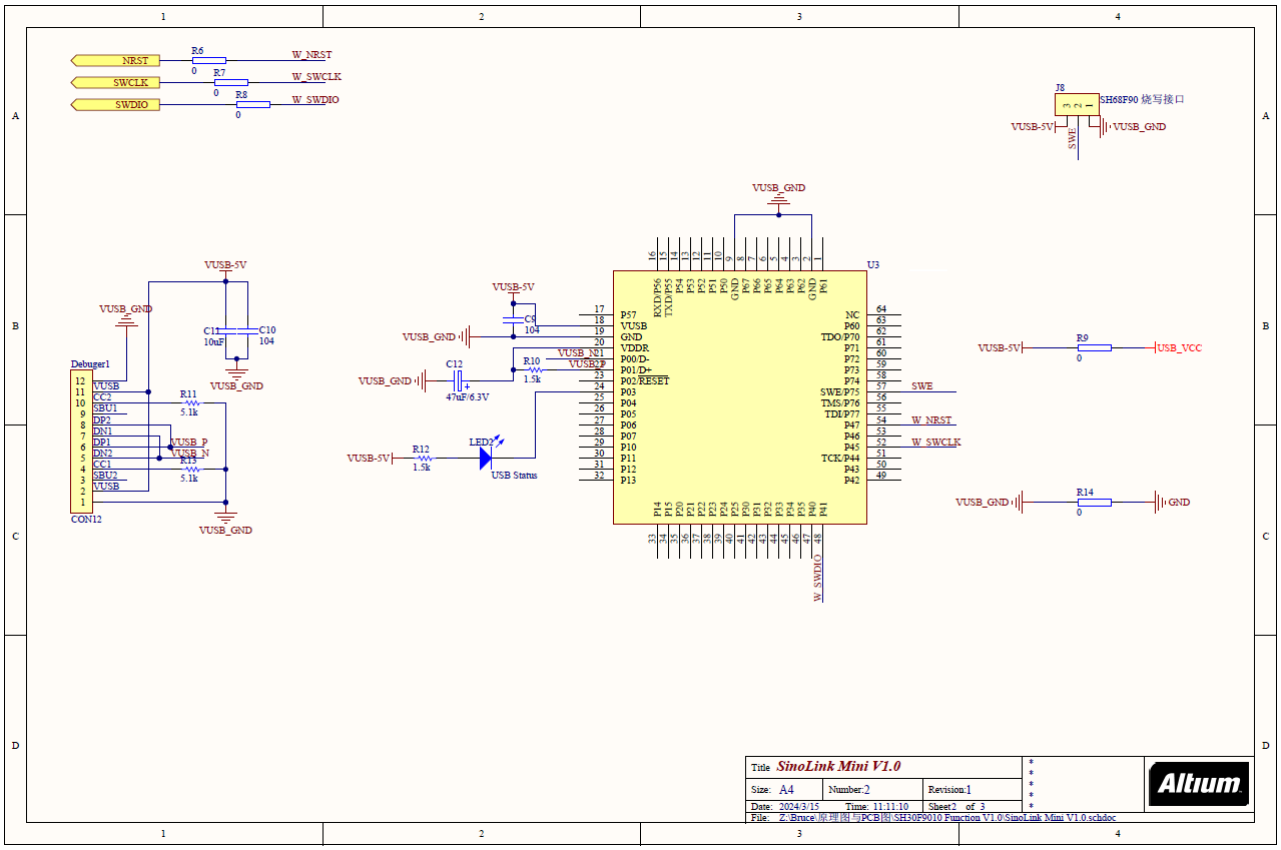
然后在 scatter file 中定义如下

```
RAM_IRAM_1 0X20000400 0x0000400{
    *(.fixed_var);
}
```

Section ".fixed_var"中的变量就被定义到了 0x20000400 起始的位置

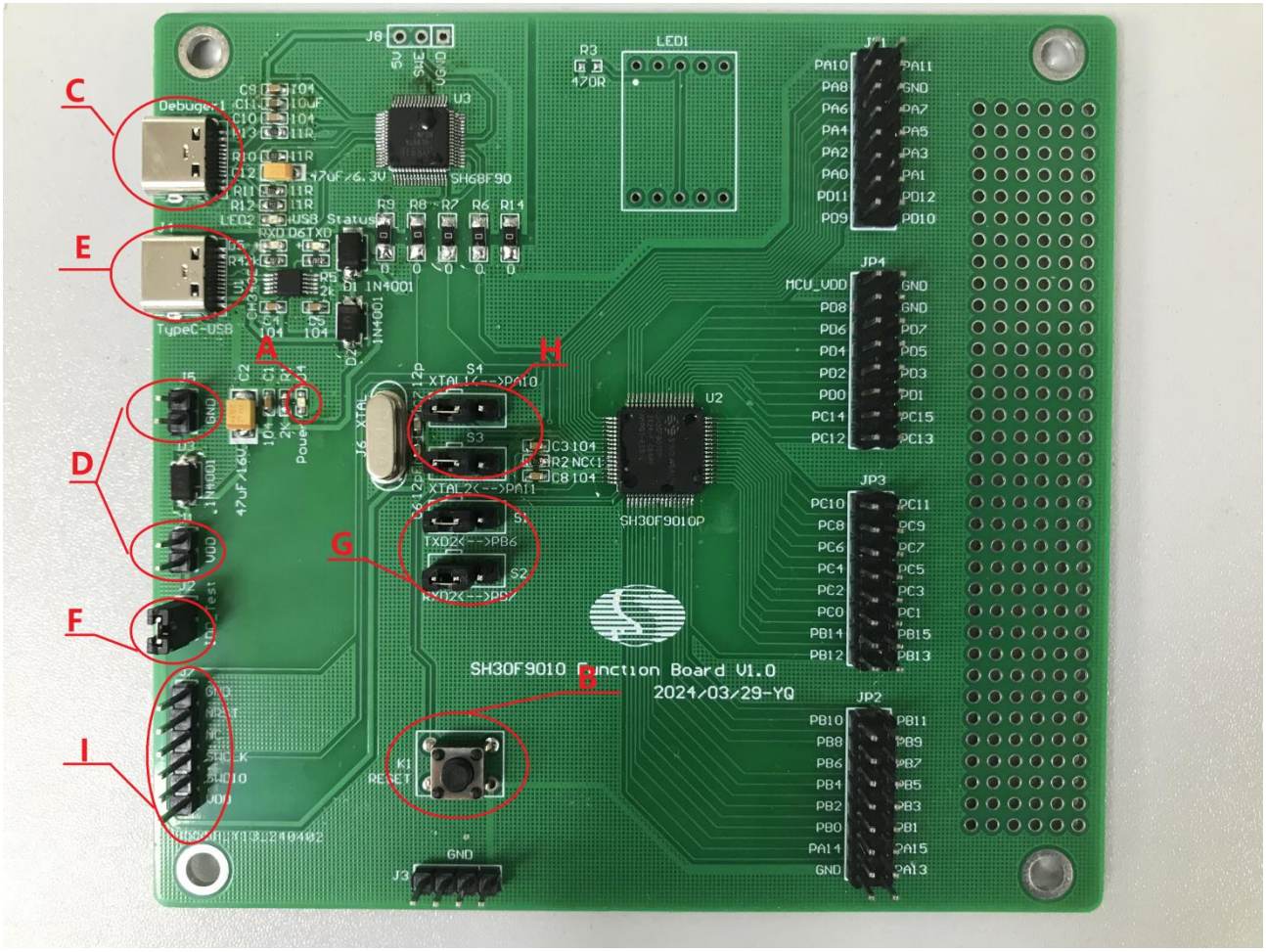
17.4. 附录 4：SH30F9010 Function Board 原理图





17.5. 附录 5：SH30F9010 演示板（Function Board）

SH30F9010 提供了一套演示板，供客户熟悉芯片功能。如下图：



SH30F9010 演示板

板中各部分功能说明：

A. 电源指示灯

B. 复位按键

C. SinoLink-Mini 调试器接口

SinoLink-Mini 板载调试器采用 USB Type-C 接口，可实现板载烧录与仿真。在配置仿真器界面中选择 CMSIS-DAP，与 jlink 一样需要添加烧写算法文件。

D. 外部供电接口（VDD/GND）

Function Board 板供电电源接口，分别与芯片的 VDD 和 GND 管脚相连，采用外部供电芯片供电电压可调。

E. USB 接口

采用 USB Type-C 接口，与 PC 连接。通过 USB 转串口芯片，连接 MCU 的 UART2(TXD2←→PB6，RXD2←→PB7) 来进行 UART 调试。

注: 演示板的供电可以由外部供电接口或 USB 接口或仿真接口提供，不管使用哪一种供电方式，外部供电接口处的电压为演示板的供电电压。

F. 电流测试接口。（不进行电流测试，请用短路跳线连接）

G. 跳线帽（UART PIN）

PB6, PB7 作为 UART2 时，跳线帽短接左边两个与板子上的 E 部分组成 USB 转串口电路；PB6, PB7 作为 IO 时，跳线帽短接右边两个，和外部的 USB 转串口电路断开。

H. 跳线帽（CRY）

PA10, PA11 作为 Crystal PIN 时，跳线帽短接左边两个和外接的晶振连接起来，PA10, PA11 作为 IO 时，跳线帽短接右边两个，断开和外部晶振的连接。

I. SWD 仿真接口

SWD 仿真接口可与 Jlink 或者 SinoLink 仿真器的 VDD、SWDIO 、SWCLK、RST、GND 使用杜邦线连接进行下载仿真。演示板的供电电源可通过仿真接口的 VDD 管脚获得。