

STM8S的配置字节OPTION BYTE

这一章节我们将详细说明一下STM8S的OptionByte。在此只以STM8S-EK开发板上的主控制芯片STM8S208R8做为目标芯片来讲解。

本章主要解决以下问题：

1. 什么是STM8S的配置字 OptionByte
2. Option Byte 里面的位都表示什么
3. 如何编程Option Byte

一 选项字节 (OptionByte)

STM8S的配置字类似于AVR的Fuse熔丝位。用于配置端口的复用功能和读保护等操作。不同于AVR的熔丝位，STM8S的时钟配置并不在Option Byte中，不会出现写完STM8S后芯片直接锁死的尴尬。

选项字节用于配置硬件特性和存储器保护状态，这些字节位于同一页的特定存储器阵列中。选项字节可以在ICP/SWIM模式中或IAP模式中修改，也就是可以通过STlink 写配置字，或者是通过程序写选项字。

当需要开启端口的第二功能时，需要考虑选项字节的编程，否则将不能实现程序所想要的功能。

例如，STM8S的有一个蜂鸣器控制器BEEP，这BEEP对应的端口是PD4.但是PD4有两个复用功能，它可以是

1. AFR7=1 BEEP蜂鸣器的输出
2. AFR7=0 Tim2 比较输出1 也就是Tim2_CC1

用户必须在这两个功能中选择一个,只能选择一个。那如何使PD4是BEEP输出呢？

答案是通过设置Option byte 选项字,将AFR7配置成1,使用PD4输出的是BEEP的信号,否则，PD4将输出的是Tim2_CC1的信号。

当然，任何的配置，用户还是可以像普通的IO一样来使用PD4。

Option Byte除了能够配置端口的复用功能外，它还可以设置芯片的读保护

当选项字节中的ROP字节被编程为'0xAA'时，读保护就生效了。这种情况下，无论写保护是否生效，在ICP模式中(使用SWIM接口)读取或修改FLASH程序存储器和DATA区域都是被禁止的。即使认为没有什么保护是完全不可破解的，对于一个通用微处理器来说，STM8的读保护的特性也提供了一个非常高水平的保护级别。

也就是说，防止别人拷贝你的程序的方法可以是编程ROP字节以使能读保护，那么 对方无论如何都不能从STM8中读到你的Flash数据(理论上)

二 选项字节详解

通过前面的介绍 你应该对OptionByte是做什么用有一个初步的认识了。那么，下面我们来讲一下，如何用OptionByte。OptionByte 具体和我的工程有什么关系呢？

仍然以STM8S208R8为例，看一下它可配置的选项字节
下图就是从STVP里截图到的

Value 00 00 00 00 00 00 00 00 00

当前值

Name	Description
ROP	Read Out Protection OFF
UBC bit7	0
UBC bit6	0
UBC bit5	0
UBC bit4	0
UBC bit3	0
UBC bit2	0
UBC bit1	0
UBC bit0	0
AFR7	Port D4 Alternate Function = TIM2_CC1
AFR6	Port B5 Alternate Function = AIN5 , Port B4 Alternate Function = AIN4
AFR5	Port B3 Alternate Function = AIN3 , Port B2 Alternate Function = AIN2 , Port B1 Al ter...
AFR4	Port D7 Alternate Function = TLI
AFR3	Port D0 Alternate Function = TIM3_CC2
AFR2	Port D0 Alternate Function = TIM3_CC2
AFR1	Port A3 Alternate Function = TIM2_CC3 , Port D2 Alternate Function = TIM3_CC1
AFR0	Port D3 Alternate Function = TIM2_CC2
LSI_EN	LSI Clock not available as CPU clock source
IWDG_HW	Independant Watchdog activated by Software
WWDG_HW	Window Watchdog activated by Software
WWDG_HALT	No Reset generated on HALT if WWDG active
EXTCLK	External Crystal connected to OSCIN/OSCON
CKAWUSEL	LSI clock source selected for AWU
PRSC	24MHz to 128KHz Prescaler
HSECNT bit7	0
HSECNT bit6	0
HSECNT bit5	0
HSECNT bit4	0
HSECNT bit3	0
HSECNT bit2	0
HSECNT bit1	0
HSECNT bit0	0
Reserved	Reserved
WAITSTATE	No wait state
BOOTLOADER ENABLE	BootLoader Disabled

读保护

用户启动区域大小

端口复用选择

低频时钟看门狗

时钟设置

时钟稳定时间

Flash等待时钟

Bootloader 使能

PROGRAM MEMORYDATA MEMORYOPTION BYTE

OPT0: ROP 读保护

这个是整个Flash的读保护使能或者是失能位，当这个字节等于 AA时，读保护生效，用户将不能读出Flash内部的信息。并不意味着这个芯片就废了，不像AVR还需要高压编程来恢复Fuse。STM8可以直接修改ROP，解除读保护，这会
引起：程序存储器、UBC、DATA区域以及选项字节都被自动擦除
这样一来，芯片就等于出厂时的设置了，器件也可以被重新编程了。

OPT1: UBC 用户启动区域大小

UBC 的全称应该是**USER BOOT CODE**，也就是用户启动代码存放的区域，可以理解成用户自己定义的Bootloader存放的地方。UBC选项字节指定了分配在UBC中的页的数量。UBC区域的起始地址是0x00 8000, UBC存放于中断向量表之后。

配置这个选项字节将决定启动区域的页数。那一页的容量有多少字节呢？

STM8和STM32 相近，很多功能都与芯片的容量有关，这里的一页多少字节也和容量有关系，

STM8分 小容量，中容量，大容量

小容量 8K FLASH程序存储器，每页 64字节，共 128页

中容量 从 16K到 32K FLASH程序存储器，每页 512字节，最多 64页

大容量 从 64K到 128K FLASH程序存储器，每页 512字节，最多 256页

比如：

UBC Bit0 =1 其他的都是0，那么组合起来，UBC =0x01；

也就是说，用户的启动代码占用1页的空间，那么对于STM8S208R8来说，这样的配置，对应的就是512 Bytes的用户启动代码空间

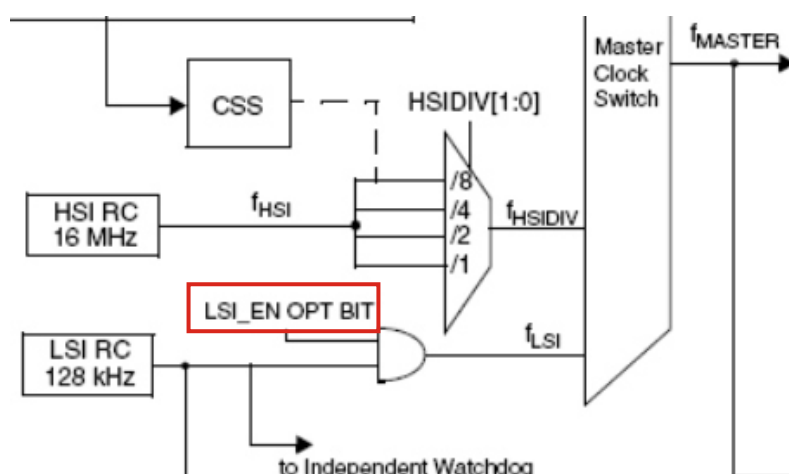
OPT2:端口复用功能

这个没有什么好说的，看着配置。按你的工程应用选择端口的第二功能。

特别提示：如果你在调一个模块的时候总是出不来信号，那么检查一下配置字节是不是选好了。

OPT3: LSI_EN低频时钟使能

这个位决定低频时钟是否能被用于系统的主时钟，请看下图, 如果LSI_EN =0那么 128K的低频率时钟将不能用于系统主时钟



OPT3:看门狗设置

IWDG_HW 决定硬件看门狗 是由硬件开启还是软件开启

WWDG_HW 决定窗口看门狗 是由硬件开启还是软件开启

WWDG_HALT 窗口看门狗溢出的时候是否产生芯片复位信号

OPT4:时钟选项

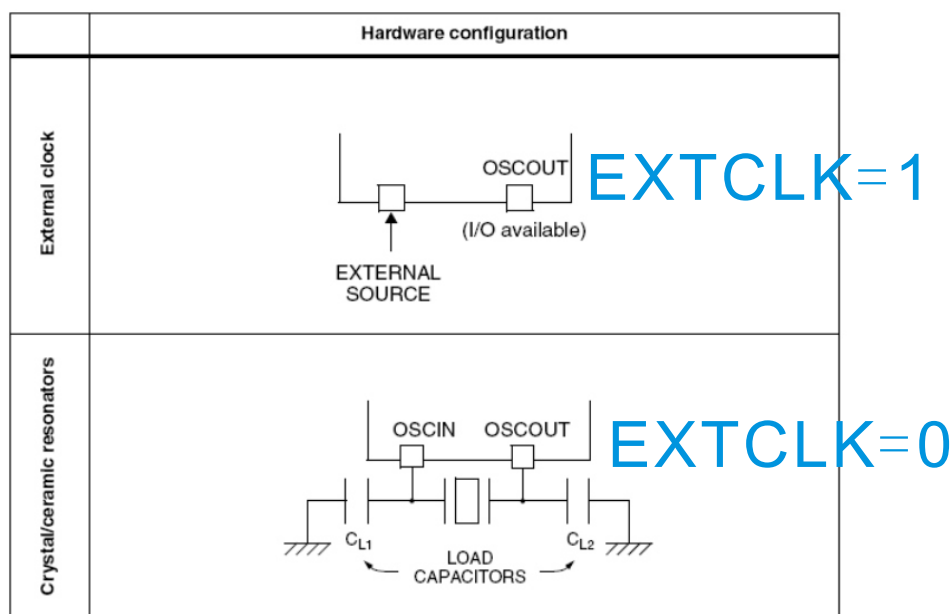
EXTCLK 外部时钟连接的硬件 两个可能：

1. 外部晶体/陶瓷谐振器 外部时钟连接的是两脚的晶体，需要芯片启动时钟
2. 外部时钟是一个信号源 输入占空比约50%的外部时钟信号(方波，正弦波，三角波)用以驱动OSCIN引脚，而OSCOUT引脚可做为通用输入/输出管脚使用。

CKAWUSEL 自动唤醒单元(AWU)的时钟源，可以选择

1. 内部低频率时钟LSI 做为AWU的时钟
2. 高频时钟HSE 经过分频后做为AWU的时钟

PRSC HSE的分频值可通过选项字节位HSEPRSC[1:0]编程。这里分出来的时钟是给AWU单元的



OPT5:HSECNT启动稳定时钟数

J外部晶体振荡器在启动时的输出时钟信号是不稳定的，默认情况下，在时钟信号被使用之前会插入2048个振荡器周期的延迟。用户可通过设置选项字节HSECNT来缩短稳定时间。也就是说，这个字节的值将决定振荡器周期的延迟，当这个字节为0x05时，那么振荡器周期的延迟将是：

2048-0x05 = 2043个周期

其他的值依此类推

OPT7: Flash等待时钟WAITSTATE

决定Flash访问是需要等待，当时钟为24M的时候，需要设置等待1个时钟。

OPTBL: BOOTLOADER ENABLE

决定系统BOOT是否启动。

在某些STM8型号中有2K字节的内部BOOT ROM，其中包含有用于启动的代码。这段代码的主要作用是利用STM8的SPI，CAN或UART接口，将应用程序代码，数据，选项字节(Option byte)和中断向量表下载到内部的FLASH和EEPROM中去。在复位以后，启动代码开始执行。

决定这段代码是否运行的条件就是这个选项字节BOOTLOADER ENABLE.

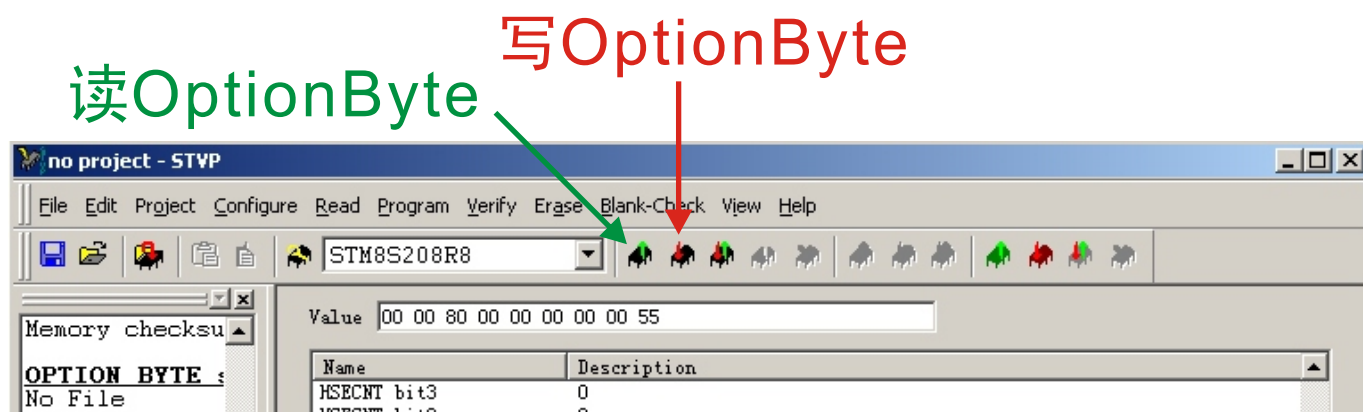
三 如何编程选项字节

好了，通过上面的学习。你已大致了解了选项字节对你的系统是多么重要，那么，我们如何编程选项字节呢？这里介绍两种方法：

1. 使用STLink 读修改写选项字节
2. 通过程序改写选项字节。

使用STlink修改选项字

这个方法无疑是最可靠最简单的，通过官方的开发工具STlink配合上位机软件STVP（全称 [ST Visual Programmer](#) ）。就可以对所有的STM8单片机进行程序的烧写和选项字节的编程。



通过程序修改选项字节

选项字节存放于数据 EEPROM中，所以我们可以通过读写EEPROM一样的操作来修改选项字节。

对选项字节编程和对DATA EEPROM区域编程非常相似。应用程序可直接向目标地址进行写操作。利用STM8的RWW功能，在对选项字节写操作的同时程序不必停下来。

Option Byte的存放地址位于0x4800-0x487F之间

建议尽量不要在程序中修改OPTION Byte 因为，假如程序跑飞产生错误的OptionByte，那对你的系统将是一个灾难。

下面是STM8S208R8的OptionByte对应的地址表。除了ROP字节外，其他的都由两个字节组成

Table 12. Option bytes

Addr.	Option name	Option byte no.	Option bits								Factory default setting
			7	6	5	4	3	2	1	0	
4800h	Read-out protection (ROP)	OPT0	ROP[7:0]								00h
4801h	Userboot code(UBC)	OPT1	UBC[7:0]								00h
4802h		NOPT1	NUBC[7:0]								FFh
4803h	Alternate function remapping (AFR)	OPT2	AFR7	AFR6	AFR5	AFR4	AFR3	AFR2	AFR1	AFR0	00h
4804h		NOPT2	NAFR7	NAFR6	NAFR5	NAFR4	NAFR3	NAFR2	NAFR1	NAFR0	FFh
4805h	Watchdog option	OPT3	Reserved				LSI _EN	IWDG _HW	WWDG _HW	WWDG _HALT	00h
4806h		NOPT3	Reserved				NLSI _EN	NIWDG _HW	NWWDG _HW	NWWDG _HALT	FFh
4807h	Clock option	OPT4	Reserved				EXT CLK	CKAWU SEL	PRS C1	PRS C0	00h
4808h		NOPT4	Reserved				NEXT CLK	NCKAWU SEL	NPR SC1	NPR SC0	FFh
4809h	HSE clock startup	OPT5	HSECNT[7:0]								00h
480Ah		NOPT5	NHSECNT[7:0]								FFh
480Bh	Reserved	OPT6	Reserved								00h
480Ch		NOPT6	Reserved								FFh
480Dh	Flash wait states	OPT7	Reserved							Wait state	00h
480Eh		NOPT7	Reserved							Nwait state	FFh
487Eh	Bootloader	OPTBL	BL[7:0]								00h
487Fh		NOPTBL	NBL[7:0]								FFh

```

/* MAIN.C file
STM8S-EK 开发板相关例程
编写者: lisen3188
网址: www.chiplab7.com
作者E-mail: lisen3188@163.com
编译环境: ST Visual Develop + STM8 Cosmic
初版时间: 2012-06-02
测试: 本程序已在第七实验室的STM8S-EK上完成测试, 实现功能见主程序main
*/
#include "STM8S208R8.h"

//OPT 数组从STVP得到
unsigned char OPT[9]={0,0,0x80,0,0,0,0,0,0x55};
void Write_OPT(void);
main()
{
    PC_DDR = 0x02;
    PC_CR1 = 0x02;
    PC_CR2 = 0x00; //LED端口输出
    Write_OPT(); //写配置字节
    PC_ODR = 0x00; //LED亮起
    while (1);
}

void Write_OPT(void)
{
    do {
        FLASH_DUKR = 0xae;
        FLASH_DUKR = 0x56;
    }
    while(!(FLASH_IAPSR & 0x08)); //解锁Flash
    FLASH_CR2 = 0x80; //对选项字节进行写操作被使能
    FLASH_NCR2 = 0x7f; //互补控制寄存器
    *((unsigned char *)0x4800) = OPT[0]; //OPT0 Read-out Protection

    *((unsigned char *)0x4801) = OPT[1]; //OPT1 User boot code
    *((unsigned char *)0x4802) = ~OPT[1]; //OPT1N Complementary of User boot code

    *((unsigned char *)0x4803) = OPT[2]; //OPT2 Alternate function remapping
    *((unsigned char *)0x4804) = ~OPT[2]; //OPT2N Complementary of Alternate function

    *((unsigned char *)0x4805) = OPT[3]; //OPT3 Watchdog option
    *((unsigned char *)0x4806) = ~OPT[3]; //OPT3N Complementary of Watchdog option

    *((unsigned char *)0x4807) = OPT[4]; //OPT4 Clock Option
    *((unsigned char *)0x4808) = ~OPT[4]; //OPT4N Complementary of Clock Option

    *((unsigned char *)0x4809) = OPT[5]; //OPT5 HSE Clock Startup
    *((unsigned char *)0x480A) = ~OPT[5]; //OPT5N Complementary of HSE Clock Startup

    *((unsigned char *)0x480D) = OPT[7]; //OPT7 Flash wait states
    *((unsigned char *)0x480E) = ~OPT[7]; //OPT7N Complementary of Flash wait states

    *((unsigned char *)0x487e) = OPT[8]; //OPTBL Bootloader
    *((unsigned char *)0x487f) = ~OPT[8]; //OPTBL Complementary of Bootloader
    //这里需要重新锁Flash, 相关代码自己写吧
}

```