TINY\_AVR 产品开发注意事项

1. FUSE

A, BOD

BODLEVEL: 根据使用的电源选择2.7V 还是4.3V或1.8V ，一般建议5V系统选择2.7V以上，3.3V系统选择1.8V以上，这个主要由整板的系统设计决定，BODLEVEL 定义的是系统工作电压，当外部电源电压低于设定值时，MCU进入RESET状态，以防止部分外设或外部信号在设定电压以下可能导致的出错。

B, WatchDog

作为产品，我们建议使用WatchDog防止由于干扰或程序缺陷导致的不可预知的后果，这个可以WDTON 来使能， 当WDTON使能后，程序要定期刷新WDT 寄存器，如果超时，MCU会被复位。

ASF 没有给出完整的watchdog驱动， 需自己添加部分驱动，以下是两种关watchdog的方式：

void WDT\_off(void)

{

#if 0

//DISABLE\_INTERRUPTS();

ENTER\_CRITICAL(UNUSED);

\_\_watchdog\_reset();

/\* Clear WDRF in MCUSR \*/

MCUSR &= ~(1<<WDRF);

/\* Write logical one to WDCE and WDE \*/

/\* Keep old prescaler setting to prevent unintentional time-out \*/

WDTCSR |= (1<<WDCE) | (1<<WDE);

/\* Turn off WDT \*/

WDTCSR = 0x00;

//ENABLE\_INTERRUPTS();

EXIT\_CRITICAL(UNUSED);

#else

MCUSR &= ~(1 << WDRF);

protected\_write\_io((void \*)&WDTCSR,

(1 << WDCE) | (1 << WDE),

(0x00 /\* Oscillator Cycles 2K \*/) | (0 << WDE /\* Watch Dog Enable: enabled \*/)

| (0 << WDIE /\* Watchdog Timeout Interrupt Enable: disabled \*/));

#endif

}

喂狗：ATmega324pb 没有watchdog Timer 寄存器，但有专门的清狗指令：

#define \_\_watchdog\_reset() \_\_asm\_\_ \_\_volatile\_\_ ( "wdr" ::: "memory")

修改watchdog 溢出周期，修改程序中WDP3，WDP2，WDP1，WDP0对应值（0/1）：

void WDT\_Prescaler\_Change(void)

{

ENTER\_CRITICAL(UNUSED);;

\_\_watchdog\_reset();

/\* Start timed sequence \*/

WDTCSR |= (1<<WDCE) | (1<<WDE);

/\* Set new prescaler(time-out) value = 64K cycles (~0.5 s) \*/

WDTCSR = (1<<WDE) | (1<<WDP3);

EXIT\_CRITICAL(UNUSED);

}

C, 在程序中设置fuse 及eeprom

Fuse 可以预先定义在程序中，如果有预先要定义的存储在EEPROM 的数据，可参考如下：

#if 1

// Example data for ATmega324PB

const char eeprdata[] \_\_attribute\_\_ ((section (".eeprom"))) =

"Hello EEPROM";

// The order of the fuse values is from low to high. 0x42 is written to Fuse byte 0, 0x81 to byte 1...

const *uint8\_t* fusedata[] \_\_attribute\_\_ ((section (".fuse"))) =

{0x42, 0x81, 0xF5};

//const char userdata[] \_\_attribute\_\_ ((section (".user\_signatures"))) =

//"Hello User Signatures";

#endif

也可以通过一下来定义FUSE

/\*for atmega4809 \*/

#include <avr/fuse.h>

// Example Fuse configuration

FUSES = {

.WDTCFG = 0x00,

.BODCFG = 0xE4,

.OSCCFG = 0x02,

.reserved\_1[0] = 0,

.reserved\_1[1] = 0,

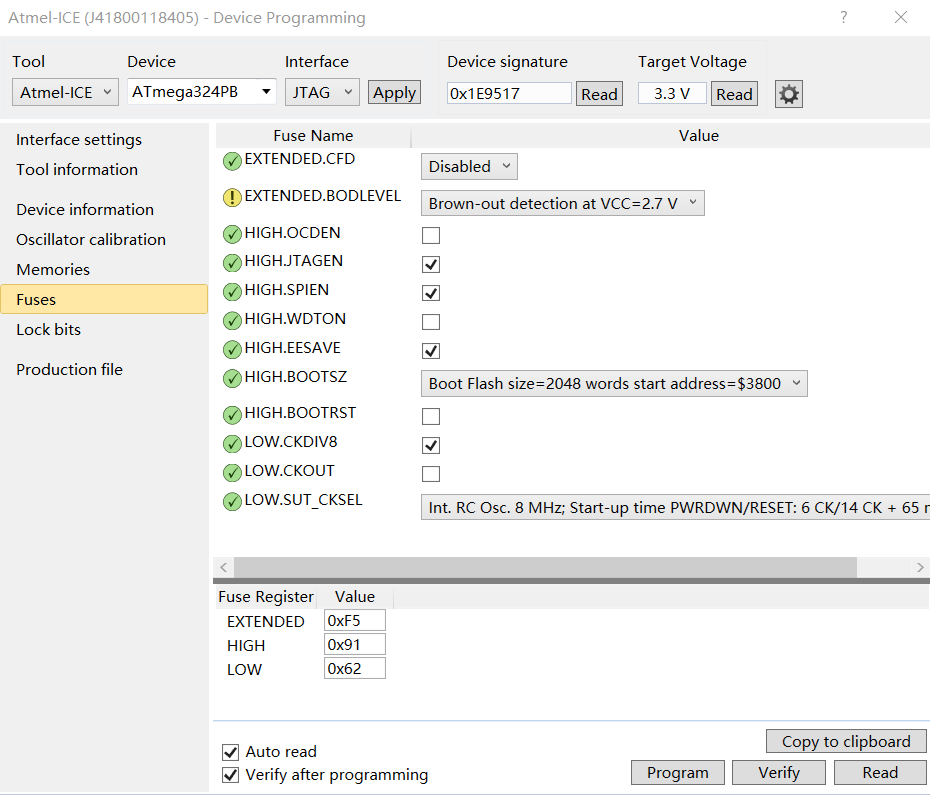
.SYSCFG0 = 0xCD,

.SYSCFG1 = 0x07,

.APPEND = 0x00,

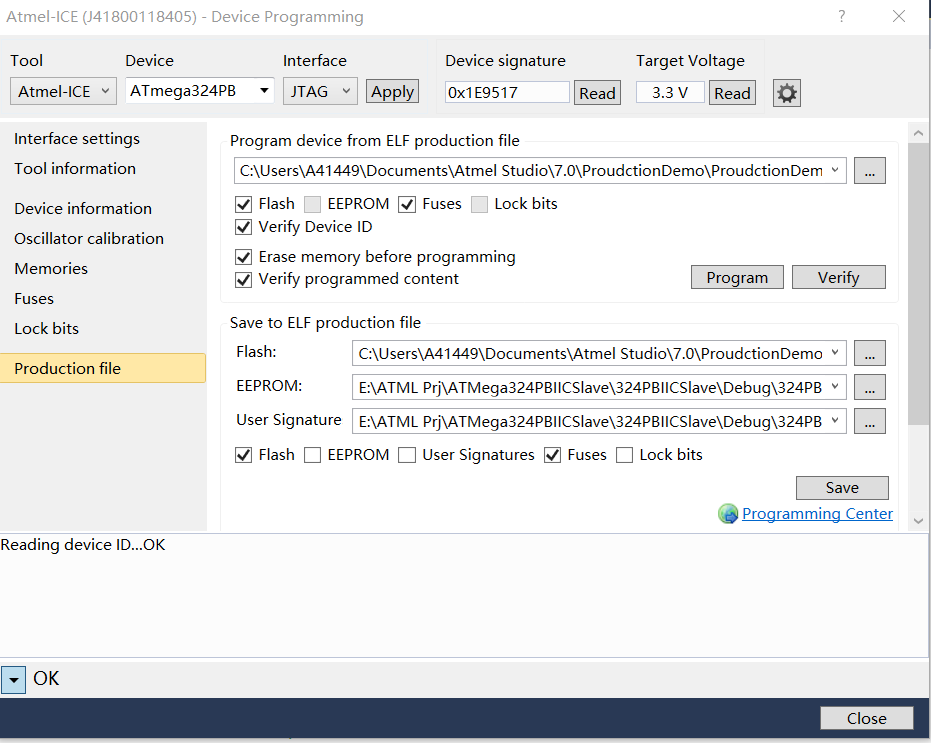
.BOOTEND = 0x00,

};



2， 量产最终目标文件

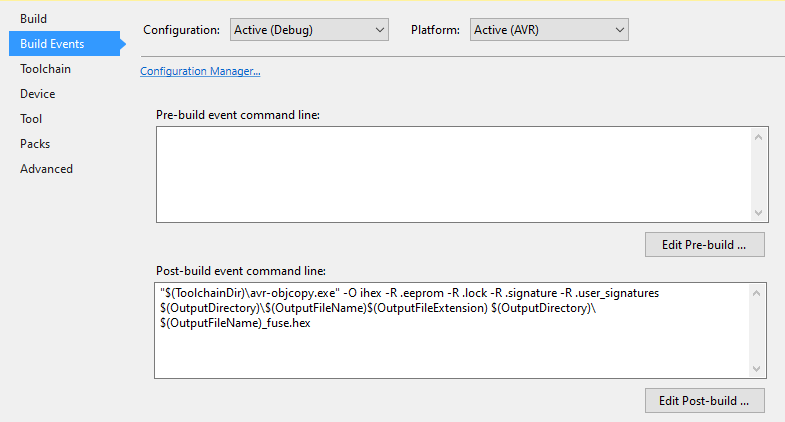
以上定义的FUSE及EEPROM 在编译后不会体现在HEX文件中， 但会存在于ELF文件中，可以用直接生成的elf文件用于量产烧写，但这个文件包含可调试信息，不利于程序的保密。可以图中Save to ELF production file 部分重新生成elf文件（包括下面的Flash，EEPROM，Fuses等选项）。



3. 生成量产HEX文件

在build Event 中加入以下命令行：

"$(ToolchainDir)\avr-objcopy.exe" -O ihex -R .eeprom -R .lock -R .signature -R .user\_signatures  "$(OutputDirectory)\$(OutputFileName)$(OutputFileExtension)" "$(OutputDirectory)\$(OutputFileName)\_fuse.hex"



其中：

avr-objcopy.exe ：

这个文件定位在 \Atmel\Studio\7.0\toolchain\avr8\avr8-gnu-toolchain\bin

-O ihex

指定输出文件格式为HEX

-R .eeprom

剔除elf文件中 .eeprom 段

-R .lock

剔除elf文件中 .lock 段

-R .signature

剔除elf文件中 . signature 段

-R .user\_signatures

剔除elf文件中 . user\_signatures 段

"$(OutputDirectory)\$(OutputFileName)$(OutputFileExtension)"

输入文件

"$(OutputDirectory)\$(OutputFileName)\_fuse.hex"

输出文件，文件名后加\_fuse ,格式为HEX

Graphical user interface, text

Description automatically generated

带FUSE 和不带FUSE 文件区别如上图：

Line53：扩展线性地址 0x0082xxxx

Line54：0x820000 共9字节FUSE 数据

注意： 转换生成的HEX文件带有FUSE信息，但要用IPE 烧写，在device programing 界面的memories 烧写是不带FUSE 信息的。

疑问： 根据datasheet 中描述， FUSE 应当定位在0x1280，但HEX中地址是0x820000， 实际由于FUSE 的地址0x1280 与代码空间的地址是重叠的， AVR 使用0x820000 映射来区分代码空间和IO空间， 这些的设置在工程中没有体现出来，在datasheet上也没找到。

如果有bootloader ，则BootLoader 编译地址为0， 应用程序link的偏移地址由下图设定

Graphical user interface, text, application, email

Description automatically generated

图例中，text 值为0x200, 但在HEX文件中，代码地址为0x400， AVR 指令为16或32位长，AVR代码地址每单元为16bit。