MICROCHIP

AN3341

AVR® DA 系列单片机的基本自举程序

简介

作者：Microchip Technology Inc.的 Cristian Pop 和 Iustinian Bujor。

本应用笔记介绍了 AVR® DA MCU 系列单片机（MCU）如何使用自编程功能。凭借自编程，用户可将应用程序代码下载到闪存，无需外部编程器。示例应用程序使用 AVR128DA48 Curiosity Nano 板，通过 UART 接口与运行 Python™脚本的 PC 进行通信。

为了避免传输无用的数据，当前实现过程会在映像的开头包含配置部分，该部分将向自举程序通知有关新映像功能的信息。此信息中包含代码的大小，因此只会将有用的数据传输到存储器中，从而显著缩短上传时间。

提供的示例自举程序和 Python™脚本适合作为自定义自举程序的起点。以下每个资源库都提供了自举程序和主机应用程序示例，此示例同时适用于 MPLAB® X 和 Atmel Studio 环境。

View the MPLAB X Projects on GitHub
Click to browse repository

View the Atmel Studio Solution on GitHub
Click to browse repository

目录

1. 相关器件

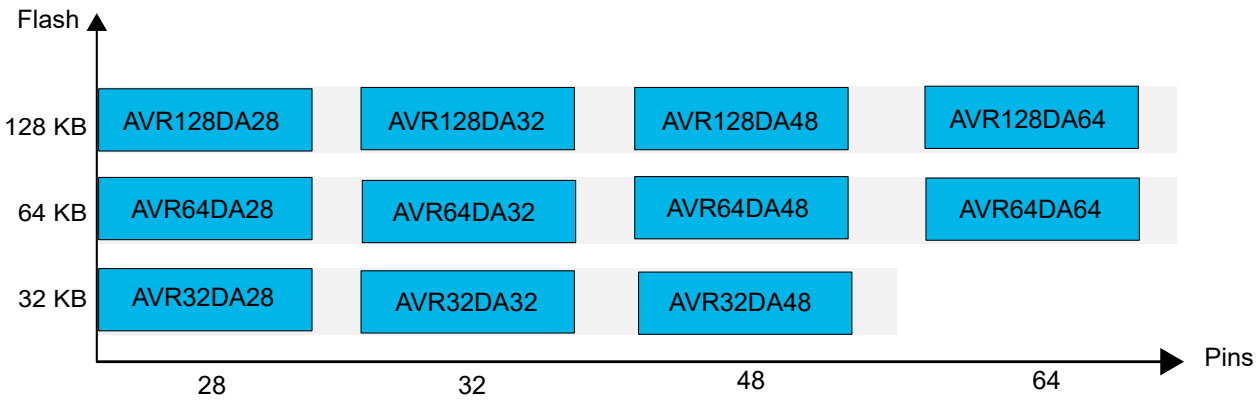
本章列出了文中涉及的相关器件。

1.1 AVR® DA 系列概述

下图给出了 AVR® DA 器件，绘制了不同引脚数型号与存储器大小的关系：

- 无需修改代码即可实现垂直移植，因为这些器件的引脚和功能完全兼容
- 水平向左移植会减少引脚数，进而减少可用的功能

图 1-1. AVR® DA 系列概述



具有不同闪存大小的器件通常也具有不同的 SRAM。

2. 器件自编程

在 AVR DA 器件上，对闪存和 EEPROM 的访问权限相比以前的 megaAVR®和 tinyAVR®器件有所更改。这意味着必须修改用于写入其他器件上的闪存和 EEPROM 的现有代码，才能在 AVR DA 器件上正常工作。本节将介绍具体的更改内容以及如何根据这些更改调整代码。

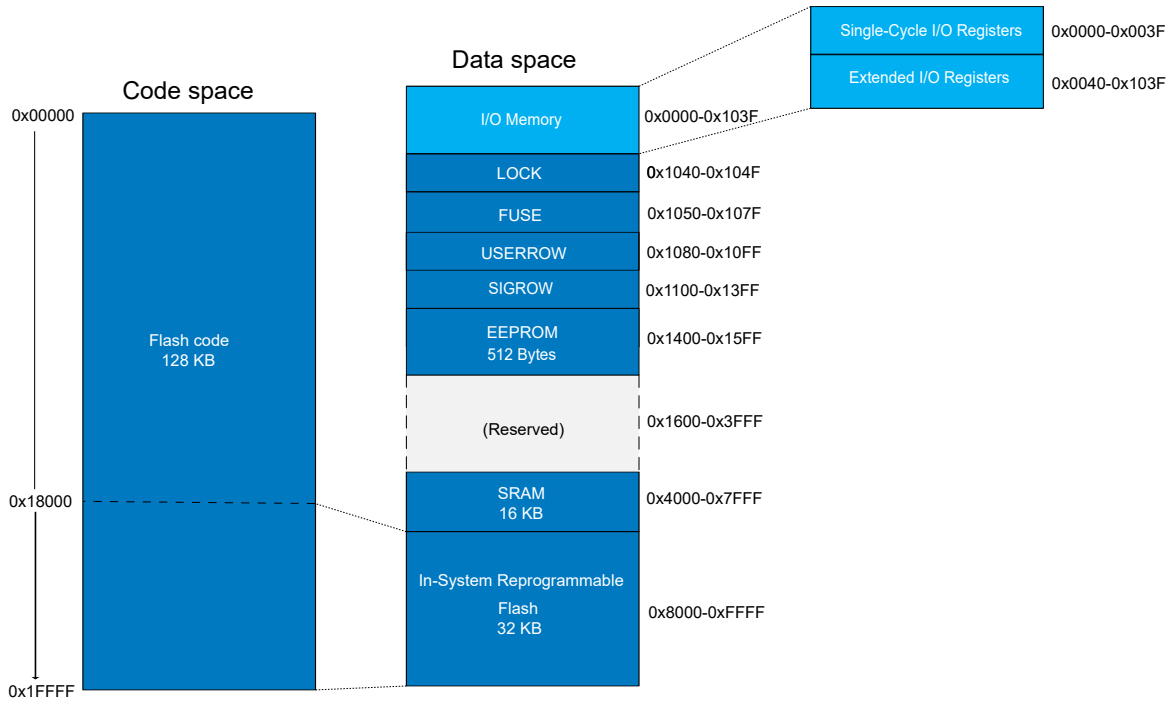
2.1 更改内容

在 AVR DA 器件上，闪存映射到代码空间中，并且可以通过 LPM 和 SPM 指令进行字访问。此外，闪存也可以映射到 CPU 数据空间中。这意味着它与 SRAM、EEPROM 和 I/O 寄存器共用相同的地址空间和指令，并且可以使用汇编语言中的 LD/ST 指令进行访问。

闪存的映射情况如下：

- 通过 LPM/SPM 指令将闪存作为程序存储器进行访问时，映射到 0x0000（PROGMEM_START）；
- 通过 LD*/ST*指令将闪存作为数据存储器进行访问时，映射到 0x8000（MAPPED_PROGMEM_START）。

图 2-1. AVR® DA 系列的存储器映射



数据空间中系统内可重复编程的闪存大小为 32 KB。对于闪存大小大于 32 KB 的器件，闪存将划分为多个 32 KB 的块。使用 NVMCTRL.CTRLB 寄存器中的 FLMAP 位域，可以将这些块映射到数据空间中：

图 2-2. CTRLB 寄存器

Bit	7	6	5	4	3	2	1	0
	FLMAPLOCK		FLMAP[1:0]			APPDATAWP	BOOTRP	APPCODEWP
Access	R/W		R/W	R/W		R/W	R/W	R/W
Reset	0		1	1		0	0	0

Bits 5:4 – FLMAP[1:0] Flash Section Mapped into Data Space
Select what part (in blocks of 32 KB) of the Flash will be mapped as part of the CPU data space and will be accessible through `LD/ST` instructions.
This bit field is not under Configuration Change Protection.

Value	Name	Mapped flash section (KB)		
		32 KB Flash	64 KB Flash	128 KB Flash
0	SECTION0	0-32	0-32	0-32
1	SECTION1		32-64	32-64
2	SECTION2		0-32	64-96
3	SECTION3		32-64	96-12

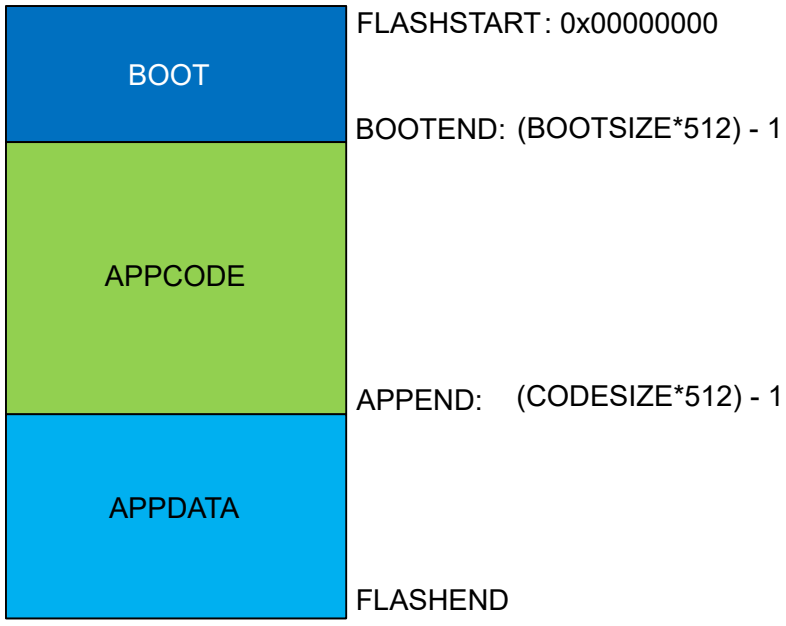
tinyAVR®和 megaAVR®器件与 AVR DA 器件之间的另一个主要区别在于对闪存的写访问。在以前的 tinyAVR 和 megaAVR 器件上，对闪存的写访问是通过页缓冲区执行的，而在 AVR DA 器件上，可以使用 `ST/SPM` 指令直接对闪存进行写访问。

注： 要写入闪存存储单元，该存储单元必须为空白（`0xFF`），否则结果为现有值与新值之间执行 `AND` 运算得到的值。

2.2 自举程序代码、应用程序代码和应用程序数据段

闪存可分为以下三个段：自举程序代码（BOOT）、应用程序代码（APPCODE）和应用程序数据（APPDATA），每个段的闪存页（512 字节的块）数可变：

图 2-3. AVR® DA 闪存映射



出于安全原因，无法将当前正在执行的代码写入闪存段。写入 APPCODE 段的代码需要从 BOOT 段执行，而写入 APPDATA 段的代码必须从 BOOT 段或 APPCODE 段执行。

熔丝定义各段对应的大小。可通过 BOOTSIZE 和 CODESIZE 熔丝控制此操作。下表列出了这些熔丝配置各个闪存段的方式。

表 2-1. 设置闪存段

BOOTSIZE	CODESIZE	BOOT 段	APPCODE 段	APPDATA 段
0	—	0 至 FLASHEND	—	—
>0	0	0 至 BOOTEND	BOOTEND 至 FLASHEND	—
>0	≤BOOTSIZE	0 至 BOOTEND	—	BOOTEND 至 FLASHEND
>0	>BOOTSIZE	0 至 BOOTEND	BOOTEND 至 APPEND	APPEND 至 FLASHEND

确保在器件上按预期设置这些熔丝的一种好方法是在自举程序代码项目中使用 `FUSES` 宏。此宏位于 `fuse.h` 中，后者通过 `io.h` 包含：

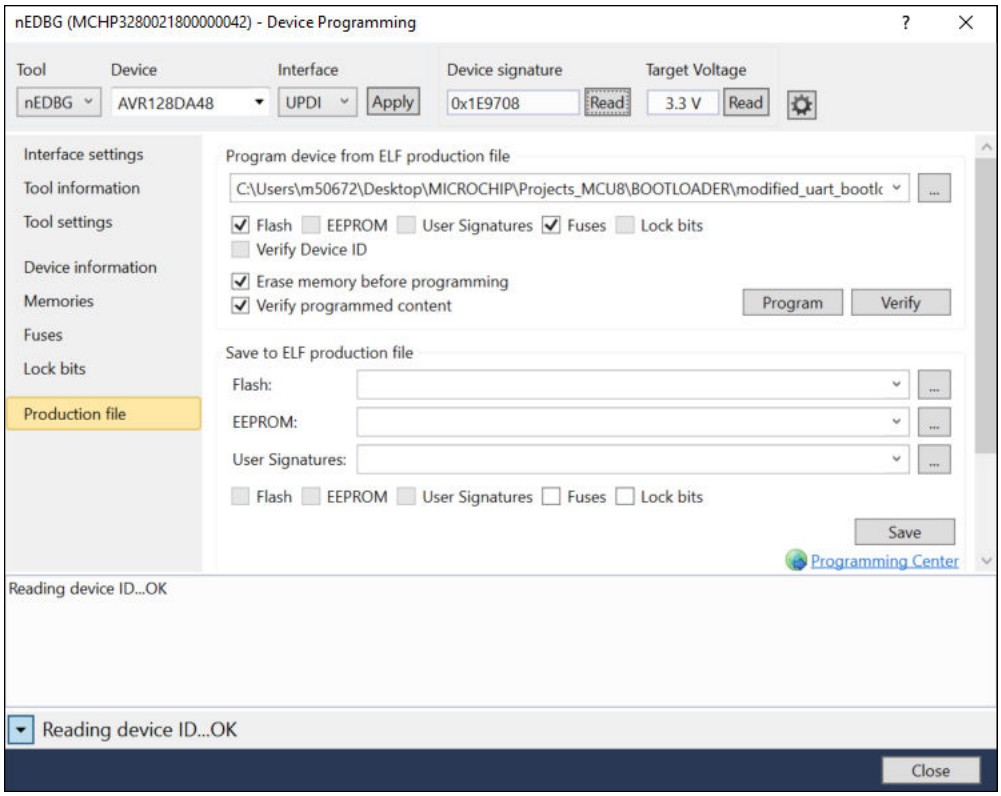
```
#include <avr/io.h>
FUSES = {
.OSCCFG = CLKSEL_OSCHF_gc,                          // 选择高频振荡器
.SYSCFG0 = CRCSRC_NOCRC_gc | RSTPINCFG_GPIO_gc,     // 未使能 CRC，RST 引脚处于 GPIO 模式
.SYSCFG1 = SUT_64MS_gc,                             // 起振时间为 64 ms
.BOOTSIZE = 0x02,                                   // BOOT 大小 = 0x02 * 512 字节 = 1024 字节
.CODESIZE = 0x00                                    // 其余所有闪存用作应用程序代码
};
```

注： 必须配置结构中的所有熔丝字节，而不仅仅是 BOOTSIZE 和 CODESIZE。原因在于，被忽略的熔丝字节将设置为 `0x00`，这可能导致意外配置。

通过在项目中包含此宏，将熔丝设置包含在编译文件中。要同时写入熔丝设置与闪存，必须在器件编程期间选择 `*.elf` 文件而不是`*.hex` 文件。

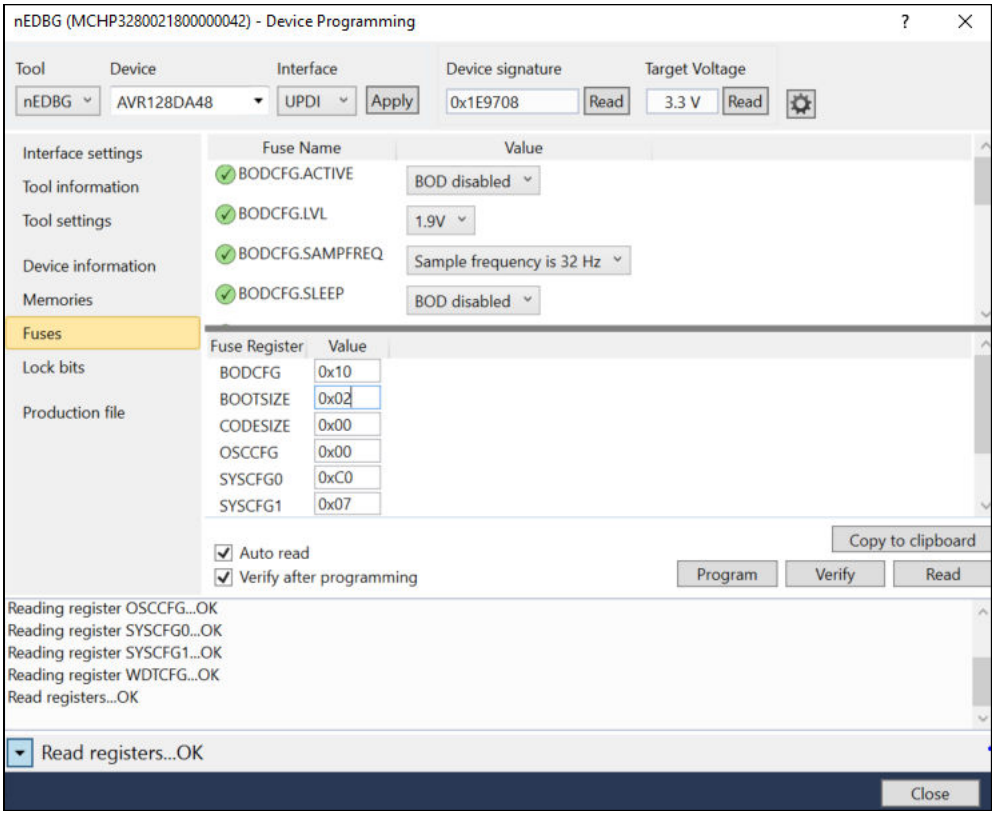
在 Atmel Studio 7.0 中，另一种方法是使用 **Device Programming**（器件编程）中的 **Production File**（生产文件）选项卡。

图 2-4. 使用 Production File 选项卡写入熔丝，Atmel Studio 7.0



在 Atmel Studio 7.0 中，也可以使用 Device Programing（**Ctrl +Shift + P**）- Fuses（熔丝）来配置熔丝，如下图所示。

图 2-5. 配置 BOOTSIZE 和 CODESIZE 熔丝，Atmel Studio 7.0



2.3 闪存编程

在 AVR DA 系列中，可通过 NVM 控制器处理闪存访问。这意味着可以使用 NVMCTRL.CTRLB 寄存器将闪存区域的各个部分（32 KB 块形式）映射到数据空间中，并且可以使用 NVMCTRL.CTRLA 寄存器来处理擦写操作。

自举程序的当前实现使用程序存储器空间来访问整个闪存。寻址是使用 `LPM/SPM` 指令完成的。

当闪存映射到程序存储器空间中时，它将被页擦除并以字粒度写入。要写入字，必须已事先擦除包含相应字地址的页，否则结果为现有值与新值之间执行 `AND` 运算得到的值。

自举程序的当前实现假定写入地址是逐步递增的，因此当写入地址输入到新页时，将通过 NVMCTRL.CTRLA 寄存器发送闪存页擦除（`FLPER`）命令。为了有效地开始擦除操作，需要对所选页执行假写操作：

```
if((addr_flash % MAPPED_PROGMEM_PAGE_SIZE) == 0x0000)                    //new page
{
                /*等待上一条命令完成*/
                while (NVMCTRL.STATUS & NVMCTRL_FBUSY_bm)
                {
                    ;
                }
                /*清除当前命令*/
                _PROTECTED_WRITE_SPM(NVMCTRL.CTRLA, NVMCTRL_CMD_NONE_gc);

                /*擦除闪存页*/
                _PROTECTED_WRITE_SPM(NVMCTRL.CTRLA, NVMCTRL_CMD_FLPER_gc);

                /*执行假写以开始擦除操作*/
                pgm_word_write(flash_addr, 0x00);
}
```

注： 要从一个命令更改为另一个命令，必须始终执行无命令（NOCMD）或无操作（NOOP）。否则，命令冲突错误将使 NVMCTRL.STATUS 寄存器中的 ERROR 位域置 1，并且将不会执行当前命令。

通过将闪存写使能（FLWR）命令写入到 NVMCTRL.CTRLA 寄存器，即可启动写入操作。在此命令之后，只要没有任何其他命令写入到 NVMCTRL.CTRLA 寄存器，就可以对所选的页存储单元进行多次字节写入操作。

```
            /*使能闪存写模式*/
        _PROTECTED_WRITE_SPM(NVMCTRL.CTRLA, NVMCTRL_CMD_FLWR_gc);

            /*将闪存字编程为所需的值*/
        pgm_word_write((flash_addr & 0xFFFFFE), data_word);
```

3. 写入自举程序

自举程序通常是一小段代码，用于对用户应用程序代码进行再编程。可使用片上通信接口（UART、I²C 和 SPI）或备用下载通道（无线接口、网络接口和外部存储器）传输新的应用程序代码。

当前示例适用于支持 AVR GNU 编译器集合（GNU Compiler Collection，GCC）的 Atmel Studio 7.0 或 MPLAB® X v5.30，此编译器集合用于编译自举程序。为了生成简短代码，将使用 AVR GCC 特定的编译器/链接器伪指令。

3.1 配置自举程序

AVR GCC 使用的标准启动文件包含中断向量表，可初始化 AVR CPU 和存储器以及跳转到 `main()`。自举程序的当前实现未使用中断，因此将删除启动文件以使代码尽可能短。

在这种情况下，不会调用 `main()`，因此需要将某个函数定义为入口点以使器件正确地开始执行。

以下代码段给出了 `boot()` 函数的示例，此函数包含需要放在 AVR GCC 代码项目的构造函数部分（`.ctors`）中的所有属性：

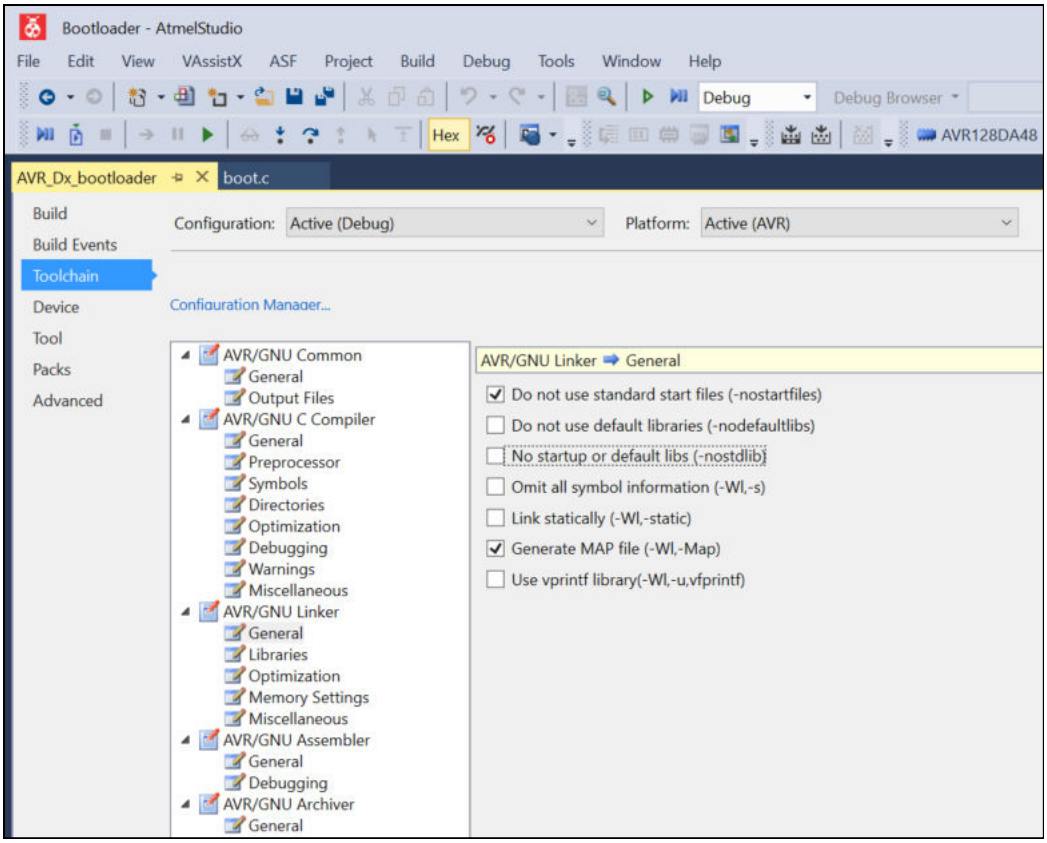
```
__attribute__((naked)) __attribute__((section(".ctors"))) void boot(void){
    /*初始化系统以便为 C 语言提供支持*/
    asm volatile("clr r1");
    /*替换为自举程序代码*/
    while (1)
    {
        ;
    }
}
```

由于未使用 `CALL/RET` 指令调用此函数而是在启动时进入了此函数，因此编译器将根据 `naked` 属性的指示省略函数的序言与结语。更多详细信息，请参阅 AVR GCC 文档。

借助 AVR GCC，在编译项目时，可通过设置链接器标志 `-nostartfiles` 禁止标准启动文件。

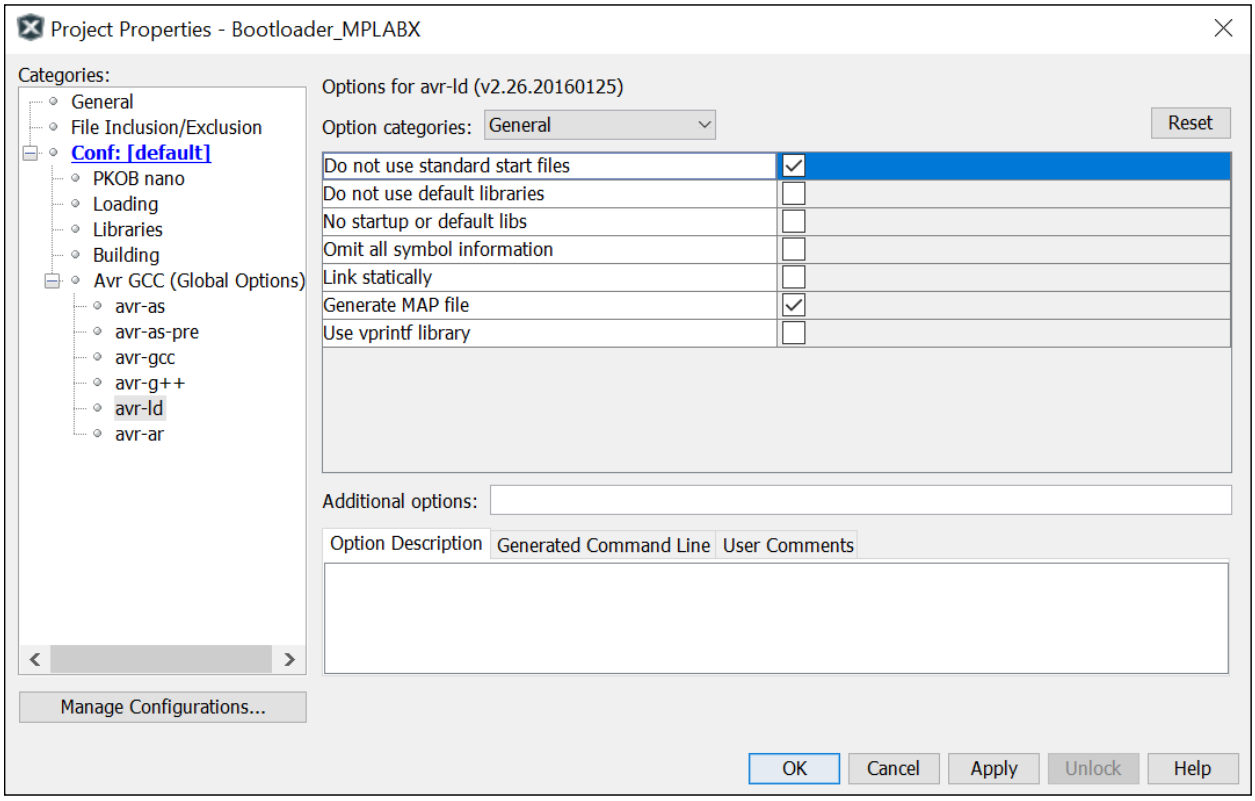
在 Atmel Studio 7.0 中，可在 *Project Properties（项目属性）（**Alt+F7**）→ Toolchain（工具链）→ AVR/GNU Linker（AVR/GNU 链接器）→ General（通用）*中找到该选项，如下图所示。

图 3-1. 禁止标准文件，Atmel Studio 7.0



在 MPLAB X 中，可在 *File（文件）→ Project Properties → Conf（配置）→ Avr GCC (Global Options)（Avr GCC（全局选项））→ avr-ld → General 中找到该选项*，如下图所示。

图 3-2. 禁止标准文件，MPLAB® X



注：

- 自举程序项目需要包括熔丝设置。更多详细信息，请参见 2.2 自举程序代码、应用程序代码和应用程序数据段。
- 生成的代码不得超过由于熔丝设置而产生的 BOOT 区域。

3.2 配置应用程序以使用自举程序

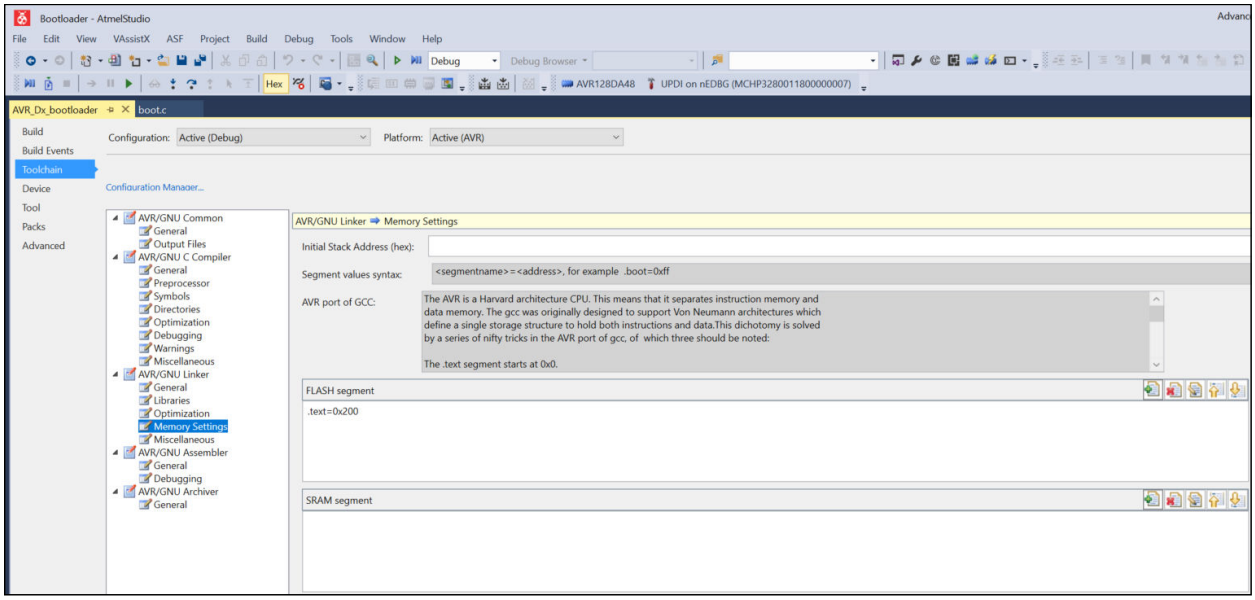
使用自举程序时，用户应用程序必须位于 BOOT 区域之后。以 BOOTSIZE 熔丝设置 `0x02` 为例，BOOT 区域的大小为 1024 字节（2 x 512 字节）。由于代码段是字对齐的，因此应用程序代码的起始地址将为 `0x200`。

为使 AVR GCC 链接器脚本了解在闪存中的哪个位置放置已编译的应用程序代码，必须将 `.text` 代码段配置为与应用程序代码段的存储单元相对应。通过使用以下链接器选项可以完成此重定位操作：

```
-Wl,--section-start=.text=0x200
```

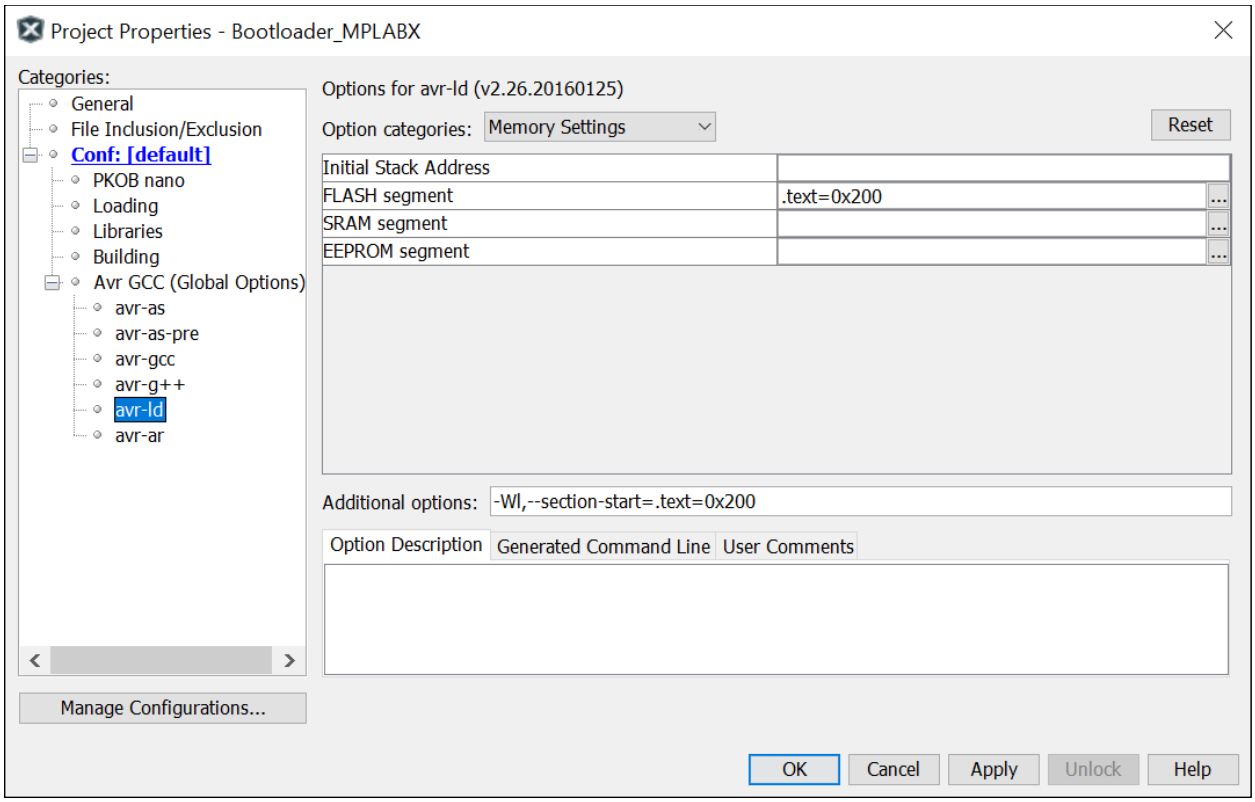
在 AtmelStudio 7.0 中，可通过在 *Project Properties (**Alt+F7**) → Toolchain → AVR/GNU Linker → Memory Settings（存储器设置）*的 FLASH segment（闪存段）中添加 `.text=0x200` 进行重定位，如下图所示。

图 3-3. 存储器设置，Atmel Studio 7.0



在 MPLAB X 中，可通过在 *File → Project Properties → Conf → Avr GCC → avr-ld → Memory Settings* 的 FLASH segment 中添加 `.text=0x200` 进行重定位，如下图所示。

图 3-4. 存储器设置，MPLAB® X



3.3 存储器保护

要保护部分段或全部闪存不被访问或写入，可通过几个保护步骤实现。惟一不能禁止的保护是闪存段写入权限，如 2.2 自举程序代码、应用程序代码和应用程序数据段所述。此外，可以配置以下类型的保护来提高安全性：

3.3.1 BOOTRP 和 APPCODEWP

引导段读保护（BOOTRP）和应用程序代码段写保护（APPCODEWP）位于 NVMCTRL.CTRLB 寄存器中，用于运行时写保护。

BOOTRP 可防止从 BOOT 段进行读访问和执行代码。该位只能通过 BOOT 段写入，并且只能通过复位清零。当 BOOTRP 置 1 时，读取 BOOT 段的任何尝试都将返回 0，从 BOOT 段执行的任何取指令操作都将返回 NOP 指令。读保护仅将在写入该位后退出 BOOT 段时才生效。

APPCODEWP 控制对应用程序代码段的写访问。当该位置 1 时，更新此段的任何尝试都将被忽略，即使从 BOOT 段执行此操作时也是如此。该位只能通过复位清零。

3.3.2 锁定位

锁定位置于单独的熔丝中，可防止编程器访问熔丝、闪存（所有自举程序代码、应用程序代码和应用程序数据段）、SRAM 和 EEPROM。

解锁器件的惟一方式是使用 CHIPERASE。不保留任何应用程序数据。

3.4 自举程序的工作原理

自举程序启动时，可以使用物理引脚状态来确定 MCU 将进入自举程序模式还是启动用户应用程序。如果该自举引脚为高电平，则将使能 BOOTRP，执行将跳转到 APPCODE；或者自举程序启动，并等待从 UART 接收数据。

进入自举程序模式时，板上 LED 会亮起，并且会在当到达页边界时熄灭。

启动后，自举程序等待通过通信接口接收附加标记（“INFO”），后跟 124 字节的数据。此 128 字节的缓冲区包含有关将要传输的应用程序映像的信息。信息缓冲区使用的结构如下：

```
typedef struct
{
    uint32_t start_mark;
    uint32_t start_address;
    uint32_t memory_size;
    uint8_t  reserved[116];
}application_code_info;
```

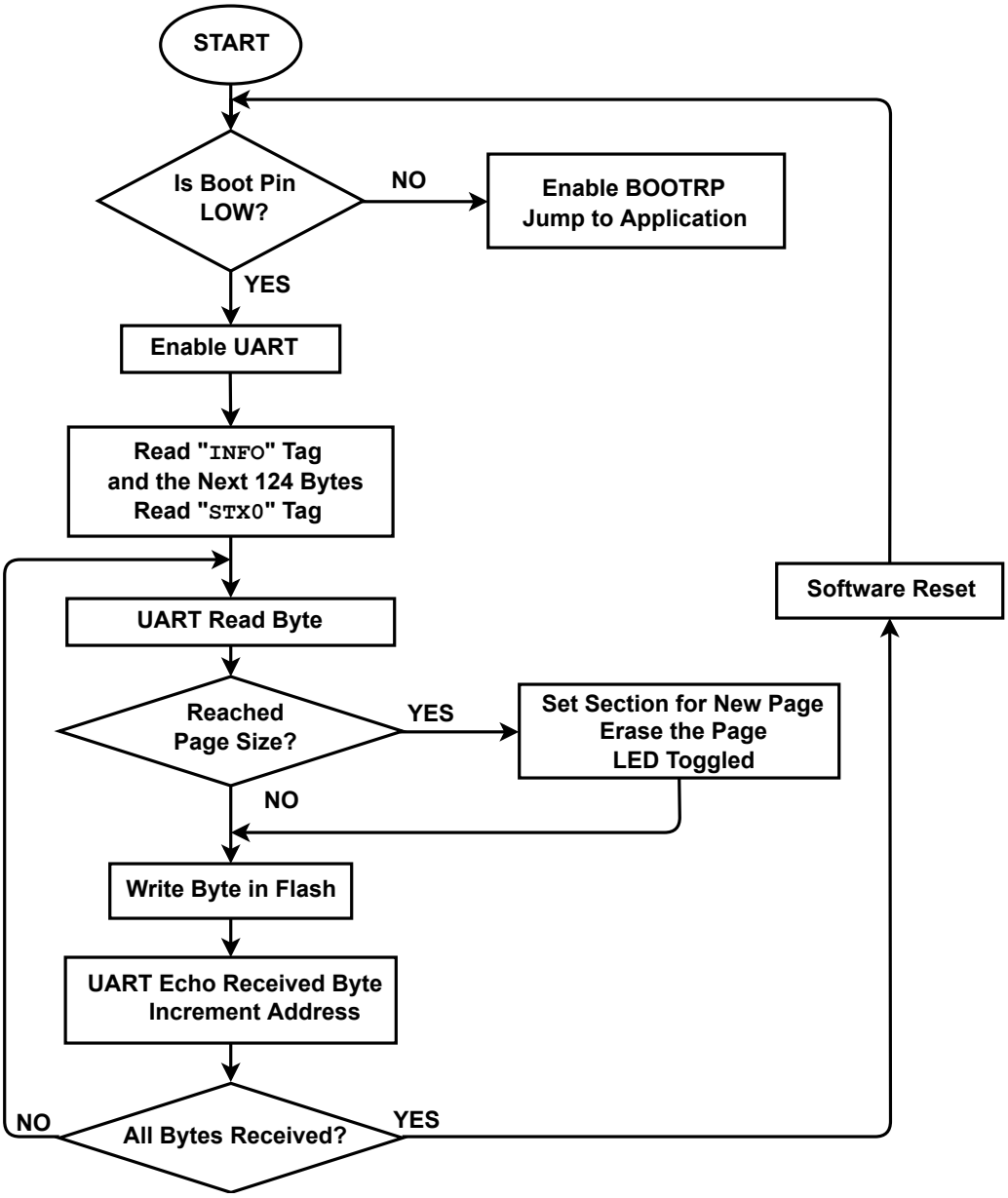
wait_mark 包含“INFO”标记；start_address 包含用户应用程序起始地址；memory_size 包含映像大小（字节数）。后面的 116 字节将保留以供未来改进之用。

接收到映像信息后，附加标记（“STX0”）预计会置于信息缓冲区的末尾，随后自举程序将开始向 APPCODE 段写入数据。

自举程序预期按照闪存地址的递增顺序逐字节接收应用程序代码。当到达新的页边界时，执行第一次擦除，并准备好进行后续写入操作。当达到映像信息缓冲区中指示的字节数时，自举程序将执行软件复位并启动新的应用程序。

下图给出了自举程序操作的流程图：

图 3-5. 自举程序流程图


START
Is Boot Pin LOW?
NO
Enable BOOTRP Jump to Application
YES
Enable UART
Read "INFO" Tag and the Next 124 Bytes Read "STX0" Tag
UART Read Byte
Reached Page Size?
YES
Set Section for New Page Erase the Page LED Toggled
NO
Write Byte in Flash
UART Echo Received Byte Increment Address
All Bytes Received?
NO
YES
Software Reset

4. 主机应用程序

在自举程序上下文中，主机是负责将应用程序代码发送到器件的系统。它通常是可以连接到目标器件以执行应用程序代码升级的计算机或 CPU，或者是同一电路板上的 CPU 主机。

对于如何创建主机应用程序的限制很少，前提是用户能够与目标器件通信。最简单的主机应用程序只有一个基本命令行接口，而更复杂的应用程序则带有图形用户界面，并且具有多层安全和高级配置设置。

如果器件具有无线连接功能，也可以进行无线（Over-The-Air，OTA）编程。这样一来，便可通过智能手机应用程序或其他升级许多器件的方式更加轻松地添加软件升级功能，而无需将每个器件物理连接到计算机或编程器。

4.1 应用程序代码格式

为了减小上传到单片机存储器中的映像大小，可以将映像信息添加到要上传的文件中。此信息包含新应用程序映像的起始地址和大小。

将通过脚本上传到单片机的.bin 文件采用如下格式：

“INFO” + 起始地址 + 应用程序大小 + 保留的字节数 + “STX0” + 应用程序代码 + 0xFF（直到页结束）。

标记“INFO”和“STX0”是通过脚本插入的，作用是通知自举程序其后面将跟随信息段，即各自的应用程序代码。

该信息段的大小为 128 字节。

由于向自举程序通知了需要写入到存储器的应用程序的大小，因此上传过程得到了简化。

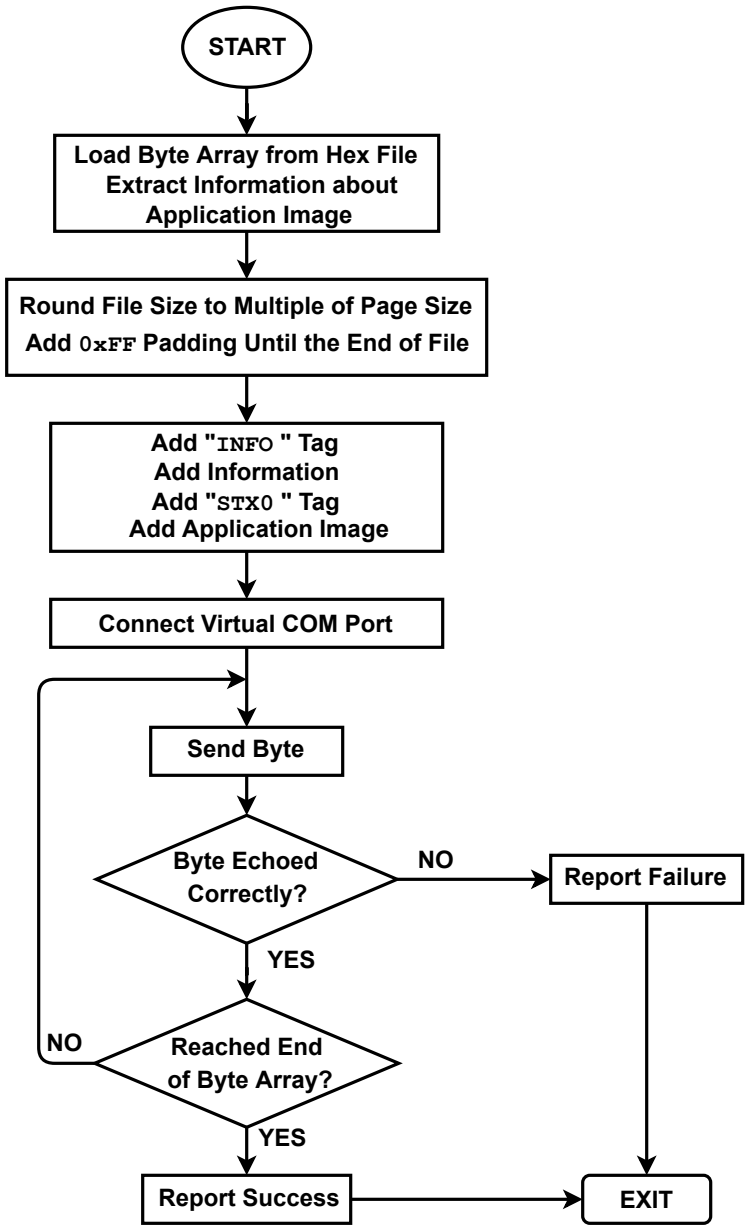
4.2 Python™脚本运行

通过提供的 Python 脚本，可以将.hex 文件转换为包含有关应用程序映像的应用程序信息（起始地址和大小）的.bin 文件。在写入信息区之前，脚本会通过在附加区域中填入 0xFF，将文件大小舍入为页大小的倍数（512 字节的倍数）。之后，将按照 4.1 应用程序代码格式中所述的结构完成信息区，并将其插入到.bin 文件的开头。

通过使用通信接口上传生成的.bin 文件，继续执行 Python 脚本。在本示例中，Curiosity Nano 板上的嵌入式调试器用作单片机与 PC 之间的桥接器。对于发送的每个字节，要求返回相同的值以确认数据传输成功。

Python 脚本流程图如下：

图 4-1. Python™脚本流程图



要运行脚本，需要以下参数：

1. 要上传的 hex 文件。如果文件不在同一文件夹中，请包含路径。
2. 单片机的最大闪存大小。
3. 用于 UART 通信的虚拟 COM 端口。
   - Windows® PC 上的 Device Manager（设备管理器）中列出了此参数。对于 AVR128DA48 Curiosity Nano，此参数以 Curiosity Virtual COM Port (COMxxx)（Curiosity 虚拟 COM 端口（COMxxx））的形式列出。在当前示例中，使用的是 COM10 端口。

图 4-2. Curiosity 虚拟 COM 端口

注： 虚拟 COM 端口连接到电路板上 AVR128DA48 器件上的 UART 引脚（PC0-TxD 和 PC1-RxD）。

4. 使用的波特率。示例自举程序使用的默认波特率为 9600。

要针对连接到端口 COM10 且闪存大小为 128 KB 的 AVR128DA48 Curiosity Nano 运行脚本，必须采用如下格式：

```
python AVR-DA_uploader.py AVR_Dx_app_example.hex 0x20000 COM10 9600
```

注： 确保在启动脚本之前将器件置于自举程序模式。这是通过在 AVR128DA48 Curiosity Nano 板上电或复位时按下 **SW0** 来实现的。

5. 扩展功能

示例自举程序是自举程序的一种简单实现，仅包含基本的功能。不过，该实现可通过多种方法扩展。本部分将介绍一些可能的扩展。

5.1 进入自举程序模式

物理引脚状态不是使器件进入自举程序模式的惟一方法。通常，应用程序必须触发自举程序更新。可以在将特定值写入用户行或 EEPROM 时触发更新。写入该特定值后，将发出软件复位命令，系统将进入自举程序模式。

5.2 通信接口

可用于主机通信的接口可能因最终应用而异。示例自举程序使用 UART 的基本配置来接收应用程序代码，这可以根据需要轻松更新，方法是替换 `boot.c` 中用作接口原型的函数。

```
/*接口函数原型*/
#define BOOTLOADER_isRequested()    BUTTON_read()
#define INTERFACE_init()            USART_init()
#define INTERFACE_readByte()        USART_read()
#define INTERFACE_writeByte(a)      USART_write(a)
```

AVR DA 系列所有器件均具有可用于串行通信的硬件 UART、TWI 和 SPI，以及可用于自定义数字协议的 I/O 引脚。

5.3 中断

自举程序的当前实现未使用中断。如果自举程序较为复杂，则可能需要使用中断。

复位后，默认的向量表位于 APPCODE 段的开头。通过将中断向量表重定位到 BOOT 段开头，可以在自举程序代码中使用外设中断。方法是将 CPUINT.CTRLA 寄存器中的 IVSEL 位置 1。

```
void relocating_vector_table_example(void)
{
    //将中断向量选择位置 1
    //读-修改-写以保留其他配置位
    _PROTECTED_WRITE(CPUINT.CTRLA, (CPUINT.CTRLA | CPUINT_IVSEL_bm));
    //允许全局中断
    sei();
}
```

注：

- 如果在自举程序中使用了中断，则不得使用链接器伪指令`-nostartfiles`。
- 如果将中断向量位置改为 BOOT 段的开头，则在进入主应用程序之前需要将其改回至 APPCODE 段的开头。

5.4 数据完整性

为确保正确接收传输到器件的代码，可以对传入数据使用循环冗余校验。这可以在接收数据时或执行代码前完成。

所有 AVR DA 器件都有循环冗余校验存储器扫描（Cyclic Redundancy Check Memory Scan，CRCSCAN）外设，可用来校验闪存内容。

5.4.1 机密性

可能需要加密对策来确保产品及其应用程序代码不会遭到克隆、伪造或篡改。在自举程序中实现 CryptoAuthentication™ 将确保只能在主机和设备之间传输合法代码。

有关更多信息，请访问 Microchip CryptoAuthentication 网站。

6. 参考资料

1. *AVR128DA28/32/48/64* 初稿数据手册。
2. *AVR128DA48 Curiosity Nano* 用户指南。

7. 版本历史

文档版本	日期	备注
C	2020 年 5 月	根据最新商标，将 AVR® MCU DA (AVR-DA)更新为 AVR® DA MCU，AVR-DA 更新为 AVR DA。
B	2020 年 3 月	更新了资源库链接。 根据最新商标，将 AVR-DA 更新为 AVR® MCU DA (AVR-DA)。
A	2020 年 2 月	文档初始版本

Microchip 网站

Microchip 网站（www.microchip.com/）为客户提供在线支持。客户可通过该网站方便地获取文件和信息。我们的网站提供以下内容：

- **产品支持**——数据手册和勘误表、应用笔记和示例程序、设计资源、用户指南以及硬件支持文档、最新的软件版本以及归档软件
- **一般技术支持**——常见问题解答（FAQ）、技术支持请求、在线讨论组以及 Microchip 设计伙伴计划成员名单
- **Microchip 业务**——产品选型和订购指南、最新 Microchip 新闻稿、研讨会和活动安排表、Microchip 销售办事处、代理商以及工厂代表列表

产品变更通知服务

Microchip 的产品变更通知服务有助于客户了解 Microchip 产品的最新信息。注册客户可在他们感兴趣的某个产品系列或开发工具发生变更、更新、发布新版本或勘误表时，收到电子邮件通知。

欲注册，请访问 www.microchip.com/pcn，然后按照注册说明进行操作。

客户支持

Microchip 产品的用户可通过以下渠道获得帮助：

- 代理商或代表
- 当地销售办事处
- 应用工程师（ESE）
- 技术支持

客户应联系其代理商、代表或 ESE 寻求支持。当地销售办事处也可为客户提供帮助。本文档后附有销售办事处的联系方式。

也可通过 www.microchip.com/support 获得网上技术支持。

Microchip 器件代码保护功能

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术规范。
- Microchip 确信：在正常使用的情况下，Microchip 系列产品非常安全。
- 目前，仍存在着用恶意、甚至是非法的方法来试图破坏代码保护功能的行为。我们确信，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这种试图破坏代码保护功能的行为极可能侵犯 Microchip 的知识产权。
- Microchip 愿与那些注重代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案（Digital Millennium Copyright Act）》。如果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

法律声明

提供本文档的中文版本仅为了便于理解。请勿忽视文档中包含的英文部分，因为其中提供了有关 Microchip 产品性能和使用情况的有用信息。Microchip Technology Inc.及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc.的英文原版文档。

商标

ISBN：978-1-5224-7346-6

质量管理体系

有关 Microchip 的质量管理体系的信息，请访问 www.microchip.com/quality。

全球销售及服务网点

美洲	亚太地区	亚太地区	欧洲
公司总部 2355 West Chandler Blvd. Chandler, AZ 85224-6199 电话：480-792-7200 传真：480-792-7277 技术支持： www.microchip.com/support 网址： www.microchip.com **亚特兰大** 德卢斯，佐治亚州 电话：678-957-9614 传真：678-957-1455 **奥斯汀，德克萨斯州** 电话：512-257-3370 **波士顿** 韦斯特伯鲁，马萨诸塞州 电话：774-760-0087 传真：774-760-0088 **芝加哥** 艾塔斯卡，伊利诺伊州 电话：630-285-0071 传真：630-285-0075 **达拉斯** 阿迪森，德克萨斯州 电话：972-818-7423 传真：972-818-2924 **底特律** 诺维，密歇根州 电话：248-848-4000 **休斯顿，德克萨斯州** 电话：281-894-5983 **印第安纳波利斯** 诺布尔斯维尔，印第安纳州 电话：317-773-8323 传真：317-773-5453 电话：317-536-2380 **洛杉矶** 米慎维荷，加利福尼亚州 电话：949-462-9523 传真：949-462-9608 电话：951-273-7800 **罗利，北卡罗来纳州** 电话：919-844-7510 **纽约，纽约州** 电话：631-435-6000 **圣何塞，加利福尼亚州** 电话：408-735-9110 电话：408-436-4270 **加拿大 - 多伦多** 电话：905-695-1980 传真：905-695-2078	**澳大利亚 - 悉尼** 电话：61-2-9868-6733 **中国 - 北京** 电话：86-10-8569-7000 **中国 - 成都** 电话：86-28-8665-5511 **中国 - 重庆** 电话：86-23-8980-9588 **中国 - 东莞** 电话：86-769-8702-9880 **中国 - 广州** 电话：86-20-8755-8029 **中国 - 杭州** 电话：86-571-8792-8115 **中国 - 香港特别行政区** 电话：852-2943-5100 **中国 - 南京** 电话：86-25-8473-2460 **中国 - 青岛** 电话：86-532-8502-7355 **中国 - 上海** 电话：86-21-3326-8000 **中国 - 沈阳** 电话：86-24-2334-2829 **中国 - 深圳** 电话：86-755-8864-2200 **中国 - 苏州** 电话：86-186-6233-1526 **中国 - 武汉** 电话：86-27-5980-5300 **中国 - 西安** 电话：86-29-8833-7252 **中国 - 厦门** 电话：86-592-2388138 **中国 - 珠海** 电话：86-756-3210040	**印度 - 班加罗尔** 电话：91-80-3090-4444 **印度 - 新德里** 电话：91-11-4160-8631 **印度 - 浦那** 电话：91-20-4121-0141 **日本 - 大阪** 电话：81-6-6152-7160 **日本 - 东京** 电话：81-3-6880-3770 **韩国 - 大邱** 电话：82-53-744-4301 **韩国 - 首尔** 电话：82-2-554-7200 **马来西亚 - 吉隆坡** 电话：60-3-7651-7906 **马来西亚 - 槟榔屿** 电话：60-4-227-8870 **菲律宾 - 马尼拉** 电话：63-2-634-9065 **新加坡** 电话：65-6334-8870 **台湾地区 - 新竹** 电话：886-3-577-8366 **台湾地区 - 高雄** 电话：886-7-213-7830 **台湾地区 - 台北** 电话：886-2-2508-8600 **泰国 - 曼谷** 电话：66-2-694-1351 **越南 - 胡志明市** 电话：84-28-5448-2100	**奥地利 - 韦尔斯** 电话：43-7242-2244-39 传真：43-7242-2244-393 **丹麦 - 哥本哈根** 电话：45-4485-5910 传真：45-4485-2829 **芬兰 - 埃斯波** 电话：358-9-4520-820 **法国 - 巴黎** 电话：33-1-69-53-63-20 传真：33-1-69-30-90-79 **德国 - 加兴** 电话：49-8931-9700 **德国 - 哈恩** 电话：49-2129-3766400 **德国 - 海尔布隆** 电话：49-7131-72400 **德国 - 卡尔斯鲁厄** 电话：49-721-625370 **德国 - 慕尼黑** 电话：49-89-627-144-0 传真：49-89-627-144-44 **德国 - 罗森海姆** 电话：49-8031-354-560 **以色列 - 若那那市** 电话：972-9-744-7705 **意大利 - 米兰** 电话：39-0331-742611 传真：39-0331-466781 **意大利 - 帕多瓦** 电话：39-049-7625286 **荷兰 - 德卢内市** 电话：31-416-690399 传真：31-416-690340 **挪威 - 特隆赫姆** 电话：47-72884388 **波兰 - 华沙** 电话：48-22-3325737 **罗马尼亚 - 布加勒斯特** 电话：40-21-407-87-50 **西班牙 - 马德里** 电话：34-91-708-08-90 传真：34-91-708-08-91 **瑞典 - 哥德堡** 电话：46-31-704-60-40 **瑞典 - 斯德哥尔摩** 电话：46-8-5090-4654 **英国 - 沃金厄姆** 电话：44-118-921-5800 传真：44-118-921-5820