AVRDUDE 中文翻译版

根据原始文档自动翻译生成，仅供学习和检索参考。命令、选项名和器件名尽量保留原文。

AVDUDE

用于下载/写入 AVR 单片机Flash、EEPROM 等的程序

适用于 AVRDUDE，版本 8.1，2026 年 4 月 5 日

作者：Hans Eirik Bull、Brian S. Dean、Stefan Rüger 和
约尔格・温施

使用 https://github.com/avrdudes/avrdude/issues 报告错误并提出问题。

版权所有©Hans Eirik Bull、Brian S. Dean、Stefan Rùger 和 Jáorg Wunsch
允许制作和分发本手册的逐字副本，前提是
所有副本均保留版权声明和本许可声明。
允许在同意的情况下复制和分发本手册的修改版本
逐字复制的版本，前提是分发整个所得的衍生作品
根据与此相同的许可通知的条款。
允许复制本手册的翻译版本并将其分发到其他语言
guage，根据上述条件进行修改版本，但本许可通知
可以在自由软件基金会批准的翻译中进行说明。

我

目录

1
简介。。。

。。。 72
附录A
平台相关信息。 73
A.1 Unix。。。

二、
A.1.5 Unix 文档。。。

1
1 简介
AVRDUDE - AVR Downloader Uploader - 是一个用于下载和上传
Atmel AVR 单片机的片上存储器。它可以对Flash、EEPROM、
并且在编程器支持的情况下，锁定位，保持单片机的熔丝位
该部件可能具有的配置和其他存储器。
AVRDUDE可以通过命令行来读取或写入芯片存储器（eeprom、
Flash、熔丝位、锁定位）和读取存储器，例如签名或校准字节；一样的
可以通过交互终端方式来实现。从命令中使用 AVRDUDE
该行非常适合根据文件内容对芯片的整个存储器进行编程，
而交互模式对于探索内存内容、修改单个字节很有用
eeprom、编程熔丝/锁定位等
对单片机进行编程要么需要一个位于两者之间的物理编程器
目标芯片和运行 AVRDUDE 的 PC，或目标上的Bootloader
然后直接连接到由 AVRDUDE 提供服务的 PC。目前，
AVRDUDE 认识 375 个零件和 203 个编程器，但不是每个编程器
可以处理每个部分。
dryrun 是一个值得注意的编程器，它允许人们
探索 AVRDUDE 命令行和终端，无需拥有或连接
真正的物理编程器。同样，dryboot 允许探索如何与
无需连接 AVR 部件即可启动加载程序。
AVRDUDE 支持以下基本编程器类型：Atmel 的 STK500、Atmel 的
AVRISP 和 AVRISP mkII 器件、Atmel 的 STK600、Atmel 的 JTAG ICE（原始的、
mkII 和 3)、appnote avr910、appnote avr109（包括 AVR Butterfly）、串行 bit-bang
适配器和 PPI（并行端口接口）。 PPI代表一类简单的程序——
其中编程线直接连接到 PC 并行端口。几个
PPI 编程器的多种变体都有引脚配置，AVRDUDE 可以
通过在上指定适当的编程器来配置与它们一起工作
命令行或通过在其配置文件中创建新条目。通常需要的所有内容
新条目的目的是告诉 AVRDUDE 每个编程功能使用哪些引脚。
多个同样简单的连接到串行端口的 bit-bang 编程适配器
也支持，其中包括流行的 Ponyprog 串口适配器和 DASA
和 uisp(1) 曾经支持的 DASA3 适配器。请注意，这些适配器是
旨在连接到物理串行端口。连接到顶部模拟的串行端口
USB 可能根本无法工作，或者工作速度非常慢。
如果您碰巧有一个至少有 4 个可用硬件 GPIO 的 Linux 系统（例如
几乎所有嵌入式 Linux 板）您无需任何额外硬件即可完成 - 只需 con-
将它们连接到 AVR SPI 接口的 SDO、SDI、RESET 和 SCK 引脚，并使用
linuxgpio 编程器类型。较旧的主板可能使用 MOSI 标签来表示 SDO 和 MISO
对于SDI。它使用 Linux sysfs GPIO 接口对线路进行 bitbang。当然，护理还是应该的
关于电压等级兼容性。另外，虽然没有严格要求，但还是强烈要求

建议以某种方式保护 GPIO 引脚免受过流情况的影响。最简单的
就是串联一些电阻，或者更好地使用三态缓冲驱动器，例如
74HC244。看看https://kolev.info/blog/2013/01/06/avrdude-linuxgpio/
有关使用此编程器类型的更详细教程。
在可直接访问 SPI 总线和 GPIO 引脚的 Linux 安装下，例如
正如在 Raspberry Pi 上发现的那样，“linuxspi”编程器类型可用于

第一章：简介
2
使用计算机上的内置接口直接连接到芯片并对其进行编程。的
使用此类型的要求是 SPI 接口与一个 GPIO 一起公开
针。由于 Linux SPI 驱动程序不保存芯片，因此 GPIO 用作复位输出
当未发生传输时选择向下，因此它不能用作复位引脚。一个
应在 SPI 总线/复位 GPIO 和
芯片，以避免在芯片损坏时可能损坏计算机的 SPI 控制器
运行电压为 5 V，SPI 运行电压为 3.3 V。可以配置选择用于复位的 GPIO
在 avrdude 配置文件中使用 linuxspi 编程器下的重置条目，或者
直接在端口规范中。之间应连接一个外部上拉电阻
AVR 的复位引脚和 Vcc。如果 Vcc 与 SPI 电压不同，则应这样做
位于电平转换器的 AVR 侧，以保护硬件免受损坏。
在 Raspberry Pi 上，接头 J8 提供对 SPI 和 GPIO 线路的访问。
通常，引脚 19、21 和 23 是 SPI SDO、SDI 和 SCK，而引脚 24 和 26 是
作为CE输出。因此，靠近这些引脚的是引脚 22 作为 GPIO25，可用作
/RESET，引脚25可用作GND。
典型的编程电缆如下所示：
J8针
ISP 引脚
名称
21
1
SDI
-
2
Vcc - 保持开路
23
3
SCK
19
4
SDO
22
5
/重置
25
6
接地
对于该编程器，-P 端口选项默认为 /dev/spidev0.0:/dev/gpiochip0。
并且，请注意 Raspberry Pi 的 3.3 V 电压电平！
STK500、JTAG ICE、avr910 和 avr109/butterfly 使用串行端口进行通信
与电脑共进晚餐。 STK600、JTAG ICE mkII/3、AVRISP mkII、USBasp、avrftdi（和
衍生品），USBtinyISP 编程器通过 USB 进行通信，使用 libusb
作为平台抽象层。 avrftdi 增加了对 FT2232C/D、FT2232H、
和 FT4232H 设备。这些都使用 MPSSE 模式，该模式具有特定的引脚映射。
位 0（配置文件中字节的 lsb）是 SCK。位 1 是 SDO，位 2 是 SDI。位

3 通常重置。 2232C/D 部件仅在接口 A 上受支持，但 H 部件
可以是 A 或 B（由 usbdev 配置参数指定）。 STK500、STK600、
JTAG ICE 和 avr910 包含板载逻辑来控制目标的编程
设备。 avr109 Bootloader实现了类似于 avr910 的协议，但实际上是
在目标Flash的引导区域中实现，而不是作为外部设备。
两种类型之间的根本区别在于用于控制的协议
编程器。 avr910协议非常简单，可以很容易地用作基础
对于一个简单的自制编程器来说，因为固件可以在线获取。另一方面
另一方面，STK500协议更加健壮和复杂，并且固件不公开
可用。 JTAG ICE 还使用类似于串行通信协议
STK500固件版本2一。然而，由于 JTAG ICE 的目的是允许片上
调试以及内存编程，协议更加复杂。 （JTAG

ICE mkII 协议也可以在 USB 上运行。）只有存储器编程功能
AVRDUDE 支持 JTAG ICE 的功能。对于 JTAG ICE mkII/3、JTAG、

第一章：简介
3
支持 debugWIRE 和 ISP 模式，前提是其固件版本至少为
4.14（十进制）。请参阅下文了解 debugWIRE 的限制。对于 ATxmega 设备，
PDI 模式（Xmega 部件）支持 JTAG ICE mkII/3，前提是它具有修订版 1
硬件和固件版本至少为 5.37（十进制）。
支持 Atmel-ICE (ARM/AVR)（JTAG、PDI、debugWIRE、ISP、UPDI）。
Atmel 的 XplainedPro 板使用 EDBG 协议（符合 CMSIS-DAP），支持
由“jtag3”编程器类型移植。
Atmel 的 XplainedMini 板使用 mEDBG 协议，也受“jtag3”支持
编程器类型。
所有模式（ISP、JTAG、PDI、HVSP、PP、debugWIRE）均支持 AVR Dragon。
当在 JTAG 和 debugWIRE 模式下使用时，AVR Dragon 的行为类似于 JTAG
ICE mkII，因此该设备的所有特定于设备的注释也将适用。使用时
在 ISP 和 PDI 模式下，AVR Dragon 的行为类似于 AVRISP mkII（或 JTAG
ISP 模式下的 ICE mkII），因此所有特定于设备的注释都将适用于此处。特别是，
Dragon 以相当快的 ISP 时钟频率开始，因此 -B 位时钟选项可能
需要实现稳定的ISP通信。对于 ATxmega 设备，AVR Dragon
在 PDI 模式下受支持，前提是其固件版本至少为 6.11（十进制）。
支持接线板（例如 Arduino Mega 2560 Rev3），利用 STK500 V2.x
协议，而是一个简单的 DTR/RTS 切换，用于将板设置为编程模式。的
编程器类型是“接线”。请注意，在这种情况下可能需要 -D 选项，
因为Bootloader会重写程序存储器，但无法真正擦除芯片
执行。
运行 STK500 1.x 协议框架的串行Bootloader通过其支持
自己的编程器类型规范“arduino”。该编程器适用于 Arduino
Uno Rev3 或任何运行 Optiboot Bootloader的 AVR。连接重试次数
attempts 可以指定为扩展参数。请参阅扩展参数部分
请参阅下文了解详细信息。
Urprotocol 是 STK500 1.x 协议的精简版本，旨在支持
病房与 STK500 v1.x 兼容；它允许Bootloader变得更小，例如
在 urboot 项目 https://github.com/stefanrueger/urboot 中实现。亲
语法类型“urclock”适合这些 urboot Bootloader。由于其落后的基础
兼容性，可以由arduino编程器提供的Bootloader通常也可以
由 urclock 编程器提供服务。这可能需要指定（到AVR-
DUDE）使用 -x bootsize=n 选项未知的Bootloader（以字节为单位），这是必要的
让 urclock 编程器能够保护Bootloader不被覆盖。如果
未知的Bootloader具有 EEPROM 读/写功能，则选项 -x eepromrw
通知 avrdude -c urclock 该功能。
BusPirate 是一款多功能工具，也可用作 AVR 编程器。单个
BusPirate 最多可连接 3 个独立的 AVR。请参阅有关扩展的部分
详细参数如下。
还支持 USBasp ISP、USBtinyISP 和 CH341A 适配器，前提是 AVR-
DUDE 已编译并支持 libusb。它们前两个具有简单的固件 -
仅USB实现，在ATmega8（或ATmega88）或ATtiny2313上运行，重新
分别地。 CH341A 编程器直接连接到 AVR 目标。他们的 SPI 位时钟
约为 1.7 MHz，无法更改。因此，AVR 目标必须

第一章：简介
4
CPU 频率为 6.8 MHz 或更高：工厂设置的 AVR 部件，通常运行
在 1 MHz 和 1.6 MHz 之间的内部振荡器上，无法使用 -c 进行编程
第 341 章。
Atmel DFU Bootloader在 FLIP 协议版本 1 (AT90USB*
和 ATmega*U* 设备），以及版本 2（Xmega 设备）。请参阅下面的一些提示
关于 FLIP 版本 1 协议行为。
JTAG、TPI 支持 MPLAB(R) PICkit 4/5/Basic 和 MPLAB(R) SNAP
ISP、PDI 和 UPDI 模式。 Curiosity Nano 板支持 UPDI 模式。它是
被称为“PICkit on Board”，因此得名 pkobn_updi。
SerialUPDI 编程器实现基于 Microchip 的 pymcuprog (https://
github.com/microchip-pic-avr-tools/pymcuprog）实用程序，但它还包含一些 per-
Spence Konde 的 DxCore Arduino 核心中包含的性能改进 (https://
github.com/SpenceKonde/DxCore）。简而言之，该编程器由简单的 USB-
转 UART 适配器、二极管和几个电阻器。它使用串行连接来提供UPDI
界面。有关更多详细信息和已知信息，请参见第 70 页第 6.3 节 [SerialUPDI 编程器]
问题。
支持 jtag2updi 编程器，可以使用 UPDI 接口对 AVR 进行编程。
Jtag2updi 只是一个可以加载到 AVR 上的固件，这使得它能够与
通过串行链接使用 jtagice mkii 协议面对 avrdude (https://github.com/
埃尔坦加斯/jtag2updi）。
协议版本 V1 和 V2 均支持 Micronucleus Bootloader。作为
Bootloader不支持从Flash读取，请使用 -V 选项来阻止
AVRDUDE 来自验证Flash。请参阅下面有关扩展参数的部分
对于微核特定选项。
所有 AVR 板均支持 Teensy Bootloader。由于Bootloader没有
支持从Flash读取，使用-V选项阻止AVRDUDE验证
Flash。有关 Teensy 特定选项，请参阅下面有关扩展参数的部分。
附录 C [编程器列表]，第 82 页，包含已知编程器的完整列表。
1.1 历史和学分
AVRDUDE 是由 Brian S. Dean 以 AVRPROG 的名义编写的，运行在
FreeBSD 操作系统。 Brian 将软件重新命名为 AVRDUDE
人们对该软件的 Windows 移植兴趣不断增长，因此该名称与
AVRPROG.EXE是Atmel的Windows编程软件的名称。
多年来，AVRDUDE 源代码驻留在 savan- 的公共存储库中
nah.nongnu.org，在那里它继续得到增强并移植到其他系统。
在
除了 FreeBSD 之外，AVRDUDE 现在还可以在 Linux、macOS 和 Windows 上运行。
的
移植工作背后的开发人员主要是 Ted Roth、Eric Weddington 和 Jorg
温施。
2022年，该项目迁移到Github（https://github.com/avrdudes/avrdude/）。
本着许多开源项目的精神，本手册还借鉴了
其他人的。最初的修订版由原始 Unix 手册页的部分内容组成
由 Jáorg Wunsch 编写，原始网站文档由 Brian Dean 编写，来自

第一章：简介
5
Brian Dean 描述 AVRDUDE 配置文件中字段的注释。的
texi 格式是根据 Ted Roth 的 Simulavr 文档建模的。

6
2 命令行选项
2.1 选项说明
AVRDUDE是一个命令行工具，使用方法如下：
avrdude -p 零件名称选项 ...
命令行选项用于控制 AVRDUDE 的行为。以下选项
被认可：
-p 零件名称
--part 零件名称
该选项告诉 AVRDUDE 哪个部分 (MCU) 连接到编程器。
Partname 参数是配置文件中列出的部件 ID。去看
当前支持的 MCU 使用列表？作为零件名称，这将为每个
部分，打印其 id；其正式零件名称；替代名称（如果有）；和名单
可用的编程接口。根据所使用的外壳，?可能需要
被引用为“?”或者 \? 。与 -v 结合使用，这还将打印一个表格
变体零件名称以及封装代码和一些绝对最大额定值。
零件 ID、其官方零件名称、列出的替代名称或任何
完整的变体部分名称可用于通过 -p 选项指定部分。如果
AVRDUDE 不知道一部分，这意味着没有配置文件条目
对于该部分，但如果您有的话，可以将其添加到配置文件中
Atmel 数据表以便您可以输入编程规范。如果-p ?
由特定编程器指定，请参见下面的 -c，然后只有那些部分
编程器期望能够处理的输出以及
可以在该组合中使用的编程接口。现实中有
可能与此列表有所偏差，特别是如果直接通过
Bootloader。请参阅附录 D [零件清单]，第 87 页，了解完整详细的信息
支持的部件列表。
-p 通配符/标志
--部分通配符/标志
运行通配符匹配的 MCU 的开发人员选项。虽然他们的
主要用途是供开发人员使用，一些标志对用户也很有用，例如 avrdude -p
m328p/S输出AVRDUDE对ATmega328P MCU属性的理解；
有关更多信息，请运行 avrdude -p x/h。
-b 波特率
--baud 波特率
覆盖相应程序中指定的 RS-232 连接波特率
mer 配置文件的波特率条目或由 default_ 定义
~/.config/avrdude/avrdude.rc 或 ~/.avrduderc 中的波特率条目
配置文件（如果没有为此编程器提供波特率条目）。
-B 位时钟
--bitclock 比特时钟
指定 JTAG、PDI、TPI、UPDI 或 ISP 间的位时钟周期
脸。
该值是以微秒为单位的浮点数。
或者，

第 2 章：命令行选项
7
该值可能会带有 Hz、kHz 或 MHz 后缀以指定位
时钟频率而不是周期。
有些编程器默认了他们的位
时钟值到1微秒位时钟周期，适合目标MCU运行
宁在 4 MHz 时钟及以上。较慢的 MCU 需要相应较高的
位时钟周期。
一些编程器将其位时钟值重置为 de-
编程软件注销时的故障值，而其他软件则存储
最后使用的位时钟值。建议始终指定位时钟
如果读/写速度很重要。您可以使用“默认位时钟”关键字
在 ~/.config/avrdude/avrdude.rc 或 ~/.avrduderc 配置文件中
分配一个默认值，以避免必须在每个
调用。
请注意，一些官方 Microchip 编程器会存储位时钟设置和
将继续使用它，直到提供不同的值。这适用于第二代
编程器（AVRISPmkII、AVR Dragon、JTAG ICE mkII、STK600）
和第三代编程器（JTAGICE3、Atmel ICE、电源调试器）。
第四代编程器（PICkit 4、MPLAB(R) SNAP）将存储最后的
用户指定的位时钟，直到编程器与计算机断开连接。
-c 编程器 ID
--programmer 编程器 ID
指定要使用的编程器。 AVRDUDE 了解不少专业人士
语法。 Programmer-id 参数是列出的编程器的 ID
配置文件。指定 -c ?列出配置文件中的所有编程器。
根据所使用的外壳，?可能需要像“? ”一样引用或者 \? 。如果你
有一个 AVRDUDE 不认识但与已知相关的编程器
编程器有机会将其添加到配置中
文件对 AVRDUDE 没有任何代码更改：复制类似的条目并更改
这些功能与未知编程器的功能不同。如果-c ?
指定了特定的部分，参见上面的-p，那么只有那些编程器是
期望能够与程序一起处理这部分的输出 -
明可以在该组合中使用的接口。现实中可以有
与此列表有偏差，特别是如果直接通过引导进行编程
装载机。请参阅附录 C [编程器列表]，第 82 页，了解完整详细的信息
已知编程器的列表。
-c 通配符/标志
--编程器通配符/标志
为通配符匹配的编程器运行开发人员选项。虽然
它们的主要用途是为开发人员提供一些标志对用户有用，例如，
avrdude -c usbtiny/S 显示了 AVRDUDE 对 usbtiny 正确性的理解
关系；有关更多信息，请运行 avrdude -c x/h。
-C 配置文件
--config 配置文件
使用指定的配置文件作为配置数据。该文件包含所有亲
AVRDUDE 知道的语法和部分定义。如果没有指定，

AVRDUDE 在以下两个位置查找配置文件：
1.应用程序加载的目录/../etc/avrdude.conf

第 2 章：命令行选项
8
2.应用程序加载的目录/avrdude.conf
如果在那里找不到，查找过程将变得依赖于平台。
开
FreeBSD 和 Linux，AVRDUDE 查看 /usr/local/etc/avrdude.conf。参见
附录A为Windows上的搜索方法。
如果配置文件写为+文件名，则在系统范围内读取该文件
和用户配置文件。这可用于将条目添加到配置中
无需修补系统范围的配置文件。可以用几个
有时，文件的读取顺序与命令行上给出的顺序相同。
-N
--无配置
不要加载通常位于的个人配置文件
~/.config/avrdude/avrdude.rc、~/.avrduderc 或在同一目录中
avrdude 可执行文件。
-A
--keep-trailing-0xff
禁用自动删除文件输入中的尾随 0xFF 序列
被编程到Flash并在 AVR 中从Flash读取。通常情况下，
尾随 0xFF 可以被丢弃，因为Flash编程需要内存
预先擦除为0xFF。 -A 应该在编程器硬件时使用，
或Bootloader软件，不执行整片擦除，而是
处理页面级别的内存擦除。流行的 Arduino Bootloader
表现出这种行为；因此，指定时默认使用 -A
-c arduino。
-D
--noerase
禁用Flash的自动擦除。当 -U 选项写入任何Flash时 -
指定了ory，avrdude将在启动任何一个之前执行整片擦除
编程操作，因为对Flash进行编程通常是一个错误
无需先执行擦除。该选项禁用该功能。自动擦除不是
用于 ATxmega 部件或 UPDI（AVR8X 系列）部件，因为它们可以
在写入每个页面之前使用页面擦除，因此不需要显式的整片擦除。
但请注意，任何不受当前操作影响的Flash页面都会
保留其先前的内容。设置-D 意味着-A。
-e
--擦除
导致执行整片擦除。这将重置Flash的内容
ROM 和EEPROM 改为值0xff，并清除所有锁定位。
除了
ATxmega和UPDI（AVR8X系列）器件，所有这些器件都可以使用页擦除，它
基本上是Flash ROM 重新编程之前的先决命令
再次。唯一的例外是如果新内容只会导致
位要从值 1 编程到 0。此选项执行芯片
在处理任何 -U、-T 或 -t 选项之前从头开始擦除。如果
在 -U、-T 或 -U 序列中的某个位置需要进行整片擦除

第 2 章：命令行选项
9
-t选项，建议使用-T擦除，而不是在
给定的命令行顺序。
在没有明确的 -e 或 -D 选项的情况下，avrdude 会尝试从 com-
命令行是否应在开始时自动擦除芯片。
如果 avrdude 检测到写入Flash的 -U 命令，则将自动擦除
在任何其他编程之前执行，除非已使用 -T 擦除命令
事先检测到，除非在写入之前读取Flash。为了追求
此分析的构成是任何终端命令都被认为可能读取Flash。
请注意，对于重新编程 EEPROM 单元，无需明确事先进行整片擦除。
需要，因为MCU在亲之前提供了一个自动擦除周期
对细胞进行编程。
-E 退出规范[,...]
--exitspecs 退出规范[,...]
将 exitspec 传递给编程器。 exitspec参数的解释
取决于编程器本身。请参阅下面的接受的编程器列表
exitspec 参数选项或发出 avrdude -E help ... 查看选项
编程器。
多个 exitspec 选项可以用逗号分隔。
-F
--力
通常，AVRDUDE 会尝试验证从设备读取的签名
在继续之前部分是合理的。因为有时会发生这样的情况
设备的设备签名已损坏（擦除或覆盖），但除此之外
正常运行时，提供此选项是为了覆盖检查。另外，对于
Atmel STK500 和 STK600 等可以调整参数的编程器
编程工具本地（独立于目标的实际连接）
控制器），此选项可以与 -t 一起使用以在终端中继续
模式。此外，尽管初始化失败，该选项仍允许继续
编程器和目标之间的连接。
-i 延迟
--isp-时钟延迟 延迟
对于 bitbang 类型的编程器，延迟大约为延迟微秒
补间每位状态变化。如果主机系统非常快，或者目标跑掉
一个慢时钟（如 32 kHz 晶体，或 128 kHz 内部 RC 振荡器），这
可能需要满足 ISP 时钟频率的要求
不得高于 CPU 时钟频率的 1/4。这是实施的
作为自旋循环延迟，甚至允许非常短的延迟。在 Unix 风格的操作上
荷兰国际集团系统，自旋循环最初是根据系统计时器进行校准的，因此
假设系统恒定，微秒数可能相当现实
AVRDUDE 运行时加载。在 Win32 操作系统上，预配置
假设每微秒的周期数可能有点偏差
非常快或非常慢的机器。

第 2 章：命令行选项
10
-l 日志文件
--logfile 日志文件
使用日志文件而不是 stderr 进行诊断输出。请注意，初始诊断
无论如何，消息（在选项解析期间）仍会写入 stderr。
-n
--测试内存
No-write：在处理 -U 时禁止将数据写入 MCU（对于
调试 AVRDUDE）。终端模式继续写入设备。
-O
--oscal
根据 Atmel 应用程序执行 RC 振荡器运行时校准
请注意 AVR053。仅 STK500v2、AVRISP mkII 和
JTAG ICE mkII 硬件。请注意，结果将存储在 EEPROM 中
地址 0 处的单元格。
-P端口
--端口 端口
使用端口来标识连接编程器的连接。
这可以是并行、串行、spi 或 linuxgpio 连接。编程器
通常指定连接类型；在没有-P规范的情况下，系统-
相关默认值default_parallel、default_serial、default_spi、
或使用配置文件中的default_linuxgpio。如果您需要使用
对于不同的端口，请使用此选项指定备用端口名称。
仅 USB 编程器通常不需要指定端口选项
因为它们是通过其供应商和产品 ID 自动识别的
avrdude.conf 或 .avrduderc。仅当有多个编程器时
插入主机的相同类型是需要 -P 选项，请参阅
下面。然而，一些 -c 编程器完全忽略 -P 选项，例如，
teensy、ch341a 或 avrftdi；这些无法区分多个插入的亲
语法。
如果avrdude配置了libserialport支持，则可以使用串行端口
使用 avrdude.conf 或 .avrduderc 中预定义的串口适配器类型指定，
例如，ch340 或 ft232r。
如果有多个相同类型的串口适配器
连接后，可以通过附加序列号来区分，例如：
ft232r：12345678。请注意，USB转串口芯片必须有一个序列号
ber 为此工作。 Avrdude 可以检查前导和尾随序列号
也匹配。在上面的示例中，ft232r:1234 也会导致
匹配，ft232r:...5678 也是如此。如果不知道USB转串口芯片
avrdude，可以使用十六进制USB供应商ID指定，十六进制
产品 ID 和可选序列号，序列号匹配后
上述规则，例如 usb:0x2341:0x0043 或 usb:2341:0043:12345678。
要查看当前插入的串行端口的列表，请使用 -P ?s。为了查看列表
avrdude 已知的所有可能的串口适配器都使用 -P ?sa。取决于
用过的外壳，？ 可能需要像“？ ”一样引用或者 \？ 。
对于 JTAG ICE mkII，如果 AVRDUDE 是使用 libusb 支持构建的，则

端口可以指定为 usb[:serialno]。在这种情况下，JTAG ICE mkII 将

第 2 章：命令行选项
11
可在 USB 上查找。如果还指定了serialno，则会匹配
从 USB 上找到的任何 JTAG ICE mkII 读取的序列号。比赛
从给定的序列号中删除任何现有的冒号后完成，并且
从右到左，因此只有序列号中的最低有效字节需要
被给予。
avrdude -v -P usb:xyz -c jtag2 -p ... 2>&1 | grep ^找到列表
所有连接到 USB 的 JTAG ICE，请参见第 2.4 节 [示例命令行
祈求]，第 28 页。
由于 AVRISP mkII 设备只能通过 USB 进行通信，因此同样
那里需要指定端口的方法。
对于运行在 HID 模式下的 USB 编程器 AVR-Doper，该端口必须是
指定为 -P avrdoper。 Unix 和 Mac OS 上需要 Libhidapi 支持
但在 Windows 上不行。有关 AVR-Doper 的更多信息，请参阅 https://
www.obdev.at/products/vusb/avrdoper.html。
对于 USBtinyISP，这是一个不实现串行数字的简单设备
bers，多个设备可以通过它们在USB层次中的位置来区分
archy 使用 -P usb:busdir:devicefile。
对于USBasp，
也可以使用 -P 来区分多个设备
usb:busdir:devicefile 或使用序列号 -P usb:serialno。的
USBasp序列号是从尾匹配的，所以只有唯一的最少的
需要有效字节。
例如，请参阅中的相应条目
附录 B [故障排除]，第 77 页。
-c pickit5 编程器允许使用十六进制覆盖产品 ID
使用 -P usb::pid 的十进制数字 pid 以及供应商和产品 ID
使用 -P usb:vid:pid 获取十六进制数字 vid 和 pid。形式-P
usb:serialno 请求 AVRDUDE 选择带有 se- 的 PICkit5 编程器
以serialno结尾的真实号码（以及配置中的vid/pid）
文件）。
对于 XBee 编程器，目标 MCU 将通过无线方式进行编程
使用 XBeeBoot Bootloader的 ZigBee 网格。
ZigBee 64 位地址
对于目标 MCU 自己的 XBee 设备，必须以 16 个字符的形式提供
十六进制值作为端口前缀，后跟@字符和串行
设备连接到与关联的第二个可直接联系的 XBee 设备
相同的网格（默认波特率为 9600）。这可能看起来类似于：
0013a20000000001dev/tty.serial。
出于诊断目的，如果带有 XBeeBoot Bootloader的目标 MCU
直接连接串口时，64位地址字段可以省略。
在此模式下，默认波特率为 19200。
对于使用某种更高级别连接到串行端口的编程器
协议（与 bit-bang 风格的编程器相反），端口可以指定为
网络：主机：端口。在这种情况下，TCP 不会尝试打开本地设备，
与主机上 (TCP) 端口的网络连接已建立。

方括号
可以放置在主机周围以提高数字 IPv6 地址的可读性
（例如，net:[2001:db8::42]:1337）。假设远程端点是
将网络流连接到本地串行的终端或控制台服务器
实际编程器所连接的端口。假设端口

第 2 章：命令行选项
12
正确配置，例如使用透明的 8 位数据连接
STK500 在 115200 波特率下无奇偶校验。
注意：IPv6 主机名和地址仅限于 Posix 系统。
-r
--重新连接
以 1200 波特率打开串口并立即关闭，等待 400 ms
对于命令行上的每个 -r，然后与
编程器。这通常称为“1200 bps touch”，用于
某些板的触发编程模式，如 Arduino Leonardo、Arduino
Micro/Pro Micro 和 Arduino Nano Every。等待时间较长，因此
对于速度较慢、功能较弱的主机，有时需要多个 -r 选项。
-q
--平息
在读取或写入时禁用（或抑制）进度条的输出
设备。再次指定它可以使操作更加安静。
-T命令
--命令cmd
轮到与其他相关的终端行 cmd -t 交互
终端、-T 终端命令和-U 内存操作。除了
最简单的终端命令，参数 cmd 最有可能需要是
用引号括起来，有关详细信息，请参阅操作系统 shell 手册。详细请看下文
所有终端命令的描述。
-t
--终端
告诉 AVRDUDE 在轮到相关终端时运行交互式终端
其他 -t 交互式终端、-T 终端命令和 -U 内存
操作。
-U 内存：操作：文件[:fmt]
--内存mem:op:fil[:fmt]
当轮到与其他 -t 间相关的操作时执行内存操作
活动终端、-T 终端命令和 -U 内存操作。内存
字段指定要操作的内存类型。从8.0版本开始内存
字段也可以是逗号分隔的存储器列表，例如 flash、eeprom；另外，
AVRDUDE 生成的 Intel Hex 或 Motorola S-Record 文件可以存储多个
美好的回忆。特殊存储器 ALL 扩展到零件具有的所有存储器
而 all 扩展到除子存储器之外的所有存储器。等是
和所有的一样；这可用于更改写入内存的顺序
到文件或从文件读取，例如签名等是所有存储器的列表，使得
签名存储器是第一位的。可以从内存中删除内存
通过在减号或反斜杠之前列出到目前为止的列表，例如 all,-calibration。使用
-T 命令行上的part选项或交互中的part命令
终端显示特定设备支持的所有内存。
通常，设备的内存配置至少包含内存类型
Flash、eeprom、签名和锁定，有时称为锁定位。

第 2 章：命令行选项
13
签名内存包含三个设备签名字节，这应该
对于该部分而言，是（但并非总是）是独特的。
一个或多个锁存储器
四个字节通常详细说明是否对Flash进行外部读/写
内存或其中的一部分是允许的。通过锁存储器限制访问后，
通常解锁内存的唯一方法是通过整片擦除。零件通常也会
有熔丝字节，它们是用于配置设备的读/写存储器
和校准存储器，通常包含只读工厂校准
价值观。
Flash在物理上实现为 NOR 存储器，其特殊之处在于
通常只能对位进行编程以将其从 1 更改为
0. 在重新编程之前，通常必须擦除Flash。
较旧的部件仅提供整片擦除功能，这也会擦除 EEPROM
除非熔丝配置保留其内容。如果 AVRDUDE 检测到 -U
写入Flash的选项可能会自动触发芯片
擦除这些旧部分。
参见-e下自动擦除的描述
上面的选项。 ATxmegas 或 UPDI 部件（AVR8X 系列）提供页擦除、
AVRDUDE 通过在编程之前擦除页面来利用这一点
除非请求 -e（整片擦除）或 -D（写入前不擦除）。
应该注意的是，在没有 -e 整片擦除选项的情况下，任何 ATxmega
或不受编程影响的 UPDI Flash页面将保留其先前的
内容。
请参阅附录 E [存储器列表]，第 96 页，了解完整的存储器列表
AVR 设备可以具有。
op 字段指定要执行的操作：
r
读取设备内存并写入指定文件
w
从指定文件中读取数据并写入设备内存
在列表中；内存列表中的只读内存会被跳过，
当编程器是Bootloader时，熔丝和锁定位；写信给
仅当内容之间存在差异时，单个只读存储器才会失败
文件和内存
v
从设备和指定文件中读取数据并执行
验证
文件字段指示要读取或写入的文件的名称。 fmt 字段是
可选，包含要读取或写入的文件的格式。可能的值
是：
我
英特尔十六进制
我
英特尔十六进制，带有读取注释和检查容差
总和错误，写入 AVR

s
摩托罗拉 S-Record
r
原始二进制文件；在Flash数据的情况下，小端字节顺序
e
ELF（可执行和可链接格式），最终输出文件
链接器；目前仅接受作为输入文件

第 2 章：命令行选项
14
米
立即模式；实际字节值在命令中指定
行，用逗号或空格分隔，代替文件字段
-U 选项。这对于编程熔丝字节非常有用，无需
必须创建单字节文件或进入终端模式。
一个
自动检测；仅对输入有效，并且仅当输入不支持时
视频位于 stdin.
d
小数；此格式和以下格式生成一行输出
put 各自的内存部分，形成一个逗号分隔的
值的列表。这对于后续的工作特别有用
处理，如熔丝位设置。
小时
十六进制；每个值都会在前面添加字符串 0x。
哦
八进制；每个值都会在前面添加 0，除非它小于 8
在这种情况下它没有前缀。
乙
二进制；每个值都会在前面添加字符串 0b。
当用作输入时，m、d、h、o 和 b 格式将使用相同的代码
阅读由空格和/或逗号分隔的数字列表。所读的
例程处理数字上的十进制、十六进制、八进制或二进制数
按数字为基础，因此数字列表可以是混合类型。在
事实上，语法与终端写入命令使用的数据相同，
即文件的输入数据也可以是2字节短整数、4字节长整数
gers 或 8 字节长整数、4 字节浮点数、8 字节双精度数
精度数字、带有终止 nul 或类似 C 字符的 C 类型字符串
例如“\t”。数字以小端方式写入内存。使用时
0x 十六进制或 0b 二进制输入前导零用于确定大小
整数的，例如 0x002a 将占用两个字节并将 0x2a 写入内存
接下来是0x00，而0x01234将占用4个字节。请参阅描述
终端写入命令了解更多详细信息。
如果没有明确的文件格式，默认情况下使用自动检测输入
文件，来自单个内存读取和英特尔的输出文件的原始二进制格式
当从存储器列表生成输出文件时带有注释的十六进制。
请注意，虽然 AVRDUDE 将从内存列表生成单个输出文件
对于除 elf (:e) 之外的所有格式，它仅识别 Intel 十六进制 (:I 或
:i)、Motorola S-Record (:s) 或 elf 文件（:e，由编译器生成）有效
读取文件以验证或写入存储器时的多存储器文件。注意事项
另外，如果文件名包含冒号作为倒数第二个字符，则 fmt 字段
不再是可选的，因为最后一个字符会被误解
作为 fmt。
当读取任何类型的Flash区域（包括各个子区域）
在 Xmega 设备中），生成的输出文件将被截断以不包含

尾随 0xFF 字节表示未编程（已擦除）内存。因此，如果
整个内存未编程，这将导致输出文件具有
根本没有内容。可以使用 -A 选项覆盖此行为。
作为缩写，形式 -U file 相当于指定 -U flash:w:file:a
或 -U application:w:file:a 对于 ATxmegas。仅当文件不存在时这才有效

第 2 章：命令行选项
15
其中有一对冒号，中间夹着一个字符，否则
第一部分可能被解释为存储器，而单个字符则被解释为存储器
操作。
用于写入Flash的文件名，以 urboot 开头：自动生成
只读 urboot Bootloader文件。尝试例如 -c dryrun -U urboot:help
了解决定Bootloader内容的功能列表。参见章节-
有关详细文档，请参阅第 5 [自动生成的文件]，第 62 页。
写作
urboot:...使用 -U Flash的文件具有也写入所需的副作用
Bootloader工作所需的所有熔丝位配置。
-v
--详细
启用详细输出。更多 -v 选项会增加详细级别。
-V
--noverify-内存
使用 -U 将数据写入 AVR 时禁用自动验证检查。
--版本
打印avrdude版本并退出
-x参数
--扩展参数
将参数作为扩展参数传递给所选的编程器实现
参数。扩展参数的解释取决于亲
语法本身。请参阅下面的接受扩展参数的编程器列表
eters 或发出 avrdude -x help ... 查看所选选项的扩展选项
编程器。该选项可以在命令行上多次使用。
-h
--帮助
显示简短的帮助文本并退出

第 2 章：命令行选项
16
2.2 编程器接受 Exitspec 参数
目前，
翻转2，
linuxspi,
LinuxGPIO,
覆盆子_pi_gpio，
pickit4_mplab，
pickit5 和老式并行端口编程器（例如 stk200 和 dapa）支持 -E
exitspec 参数选项。
这些让用户决定编程器处于哪种状态
编程会话结束后引脚。 AVRDUDE 只允许一个 -E 选项。然而，
可以将多个 exitspec 参数指定为一个以逗号分隔的列表。
翻转2
linuxspi
linuxgpio
树莓派_GPIO
并行端口编程器
帮助
显示帮助菜单并退出。
重置
‘/RESET’信号将在程序退出时保持激活状态，即
是否将保持低电平，以保持 MCU 处于复位状态
之后。特别注意的是，编程算法
AT90S1200 器件要求“/RESET”信号处于活动状态
在给 MCU 上电之前，所以如果外部电源是
用于此 MCU 类型，之前调用 AVRDUDE
指定的此选项是保证这一点的可能方法之一
条件。如果出现以下情况，flip2 将不会在程序退出时退出Bootloader模式：
使用重置。
诺雷塞特
‘/RESET’行将在程序退出时停用，因此
编程时降低MCU目标程序运行
硬件保持连接。 Flip2 将退出Bootloader模式
如果使用 noreset 则程序退出并启动应用程序，并且这
是此Bootloader的默认行为。
并行端口编程器
电压控制电路
此选项将使这些并行端口引脚保持活动状态（即高电平）
可用于向 MCU 提供“Vcc”电源。
诺瓦克
此选项会将并行端口的“Vcc”引脚拉低
程序退出。
d_高
此选项将使并行端口上的 8 个数据引脚保持活动状态
（即高）。

d_低
此选项将使并行端口上的 8 个数据引脚保持不活动状态
（即低）。
Pickit 4 (MPLAB)
皮克特 5
电压控制电路
该选项将使编程器不再供电
avrdude 完成操作后即可。默认禁用。

第 2 章：命令行选项
17 号
2.3 编程器接受扩展参数
扩展参数是编程器特定的选项；它们都以 -x 开头。一般来说，
每个编程器都会允许 -x help，这将显示已知扩展的帮助菜单
该编程器的参数（如果有），然后退出。
下面的扩展参数是
所有显示都没有必要的 -x 选项导入。 AVRDUDE 允许任意数量的 -x
要在命令行上指定的扩展参数。
试运行
干靴
Dryrun 模拟外部编程，无需连接专业设备
语法或一部分，而 dryboot 模拟Bootloader编程，无需
需要连接目标部分。他们接受以下参数：
初始化
用人类可读的模式初始化内存。Flash
将随机配置Bootloader、数据和
代码长度。使用固定宽度字体可以最好地看到图案
:I 格式通过检查生成的十六进制文件或使用，
例如，-U flash:r:-:I。Flash中的模式是可执行的并且
代表良性 AVR 代码，即无 I/O 内存访问。选择一个
固定种子以获得可重复的结果。
初始化=n
-x init -x seeds=n 的快捷方式（见下文）
随机的
使用随机代码和值初始化存储器。Flash
将随机配置Bootloader、数据和
代码长度。
Flash中的随机代码将是良性的，即不是
访问 I/O 存储器、SRAM 或Flash。选择固定种子
可重复的结果。
随机=n
-x 随机 -x 种子=n 的快捷方式
种子=n
n 的种子随机数生成器；默认为时间（NULL）。
使用固定的正 n 设置此选项将使随机
选择可重复，即它们在不同的情况下保持相同
阿夫杜德跑了。
JTAG ICE mkII/3
爱特梅尔-ICE
PICkit 4
MPLAB(R) SNAP
电源调试器
AVR龙
使用 JTAG ICE mkII、JTAGICE3 时，
爱特梅尔-ICE、PICkit 4、
MPLAB(R) SNAP、电源调试器或 AVR Dragon 处于 JTAG 模式，

接受以下扩展参数：
jtagchain=UB,UA,BB,BA
为之前的 UB 单元、之后的 UA 单元设置 JTAG 扫描链，
分别是目标 AVR 之前的 BB 位和之后的 BA 位。
链中的每个 AVR 单元移位 4 位。
其他JTAG
单位可能需要不同的位移计数。

第 2 章：命令行选项
18
胡布迪
仅限 Power Debugger 和 Pickit 4
高压 UPDI 编程用于启用 UPDI 引脚
之前已设置为 RESET 或 GPIO 模式。使用 -x hvupdi
为支持的目标启用高压 UPDI 初始化。
vtarg=值，vtarg
仅电源调试器
可以通过设置目标电压来启用电压发生器。
当前设置电压可以单独通过 -x vtarg 读取。
PICkit 4
MPLAB(R) SNAP
PICkit 4 和 MPLAB(R) SNAP 编程器接受以下 ex-
倾向参数：
模式=avr
将编程器切换到 AVR 模式，然后退出，除非它已经
在 AVR 模式下
模式=<mplab|图片>
将编程器切换到 MPLAB aka PIC 模式，然后退出
PICkit 4 和 MPLAB(R) SNAP 编程器只能由 AVR 使用
DUDE 在 AVR 模式下。使用 -x mode=avr 切换到 AVR 模式，或者
-x mode=mplab 用于切换（返回）到 MPLAB 模式。
PICkit 5
PICkit 4（PIC 模式）
PICkit 5 和 PICkit 4（MPLAB 模式）编程器可接受以下
扩展参数
vtarg=值
指定编程器可以使用的 1.8 至 5.5 V 之间的电压
应该供应给目标。
如果已经存在有效电压
应用于 VTG 引脚时，该设置将被忽略。
当
AVRDUDE 检测到外部电压超出此范围，它
将终止操作。您可以通过设置禁用此检查
电压至 0 V。如果选择了 XMEGA 部件，则需要
电压高于 3.49 V 将导致操作中止。通常，
当会话结束时，编程器将停止供电。
要继续为目标供电，您可以使用 -E vcc 选项。
胡布迪
高压 UPDI 编程用于启用 UPDI 引脚
之前已设置为 RESET 或 GPIO 模式。使用 -x hvupdi
为支持的目标启用高压 UPDI 初始化。
根据目标，高压脉冲将施加在
RST 引脚或 UPDI 引脚。
Xplained 迷你
Xplained Mini/Nano 编程器（ISP 或 UPDI，而非 TPI）类型接受

以下扩展参数：

第 2 章：命令行选项
19
受苦=VALUE，受苦
SUFFER 寄存器允许用户修改
板载 mEDBG。当前状态可以通过-x受苦来读取
独自一人。
位 7 Arduino：
设置为 0 时添加额外 LED 的控制
位 6..3：保留（必须设置为 1）
位 2 EOF：
主动断电，如果没有 USB，则 5 秒后休眠
设置为 0 时的枚举
位 1 低：
当位设置为 0 时，强制以 1 MHz 运行 mEDBG
位 0 熔丝位：
当位发送到 1 时，熔丝位被安全屏蔽。
设置为 0 时不受保护
vtarg_switch=值，vtarg_switch
板载目标电压开关可以通过以下方式打开或关闭
写入1或0。当前状态可以通过-x vtarg_读取
单独切换。请注意，目标电源开关将始终处于
电源循环后亮起。另请注意，较小的 Xplained Nano
主板没有目标电源开关。
好奇号纳米号
Curiosity Nano 板接受以下扩展参数：
vtarg=值，vtarg
生成的板载目标电压可以通过指定来更改
新的电压。当前设置电压可以通过-x vtarg读取
独自一人。
STK500
STK600
STK500 和 STK600 板接受以下扩展参数：
vtarg=值，vtarg
生成的板载目标电压可以通过指定来更改
新的电压。当前设置电压可以通过-x vtarg读取
独自一人。
fosc=值[MHz|M|kHz|k|Hz|H], fosc
设置可编程振荡器频率，单位为 MHz、kHz 或 Hz。的
当前频率可以单独通过-x fosc读取。
varef=值，varef
生成的板载模拟参考电压可以通过以下方式更改
指定新的参考电压。当前参考电压
可以单独通过 -x varef 读取。

第 2 章：命令行选项
20
varef[0,1]=值, varef[0,1]
仅限 STK600
为通道 0 或生成的板载模拟参考电压
可以通过指定新的参考电压来更改通道 1。
当前参考电压可以通过-x varef0或-x读取
单独的 varef1。
尝试[=1..99]
仅STK500V1
指定在执行之前执行的连接重试尝试次数
伊廷。如果未指定，则默认为 10。
xtal=值[MHz|M|kHz|k|Hz|H]
定义编程器的 XTAL 频率（如果它不同于）
原始 STK500 的 7.3728 MHz。由 avrdude 用于cor-
fosc 和 sck 的直接计算。
AVR109
AVR109 编程器类型接受以下扩展参数：
自动重置
在端口打开时切换 RTS/DTR 线路以发出硬件重置。
AVR910
Atmel 低成本 AVR910 编程器类型接受以下扩展
参数：
开发代码=值
使用 VALUE 作为设备来覆盖设备代码选择
代码。
不会向编程器询问支持的列表
设备代码，指定的VALUE未验证但已使用
直接在T命令内发送给编程器。价值
可以使用 C 的传统数字表示法来指定
编程语言。
无块模式
禁用块传输功能的默认检查。使用不_
仅当您的 AVR910 编程器在执行期间产生错误时才采用块模式
初始序列。
Arduino
Arduino 编程器类型接受以下扩展参数：
尝试[=1..99]
指定在执行之前执行的连接重试尝试次数
伊廷。如果未指定，则默认为 10。
不自动重置
不要在端口打开时切换 RTS/DTR 线路，以防止硬件损坏
重置。
乌尔时钟

第 2 章：命令行选项
21
urclock 编程器类型接受以下扩展参数：
展示
显示连接部分的所有信息，然后退出。 -x 显示...
以下选项可用于组装定制响应，其中包括
所有可用相关信息的子集（或仅一项）
关于连接部分和Bootloader。
显示ID
显示存储在Flash或 EEPROM 中的唯一 Urclock ID
MCU，然后退出。
id=E|F.addr.len
从历史上看，Urclock ID 是一个六字节的唯一小端数字
ber 存储在 Urclock 板的 EEPROM 地址 257 处。
该数字的阳离子可以通过 -x id=E|F.addr.len ex- 设置
倾向参数。 E 代表 EEPROM，F 代表Flash。一个
负地址addr从EEPROM和Flash末尾开始计数，
分别。 Urclock ID 的长度 len 可以在 1 之间
和8个字节。
演出日期
显示 Flash 应用程序输入文件的最后修改日期
化，然后退出。如果输入文件是 stdin，则日期将为
编程。日期和文件名是元数据的一部分
urclock 编程器默认存储在高位Flash中
Bootloader；另请参见 -x nometadata。
显示文件名
显示最后一次Flash写入会话的输入文件名（或标题），
然后退出。
标题=字符串
设置后，将使用字符串代替输入文件名。的
标题/文件名字段的最大字符串长度为 254 字节 -
包括终止 nul。
显示应用程序
显示已编程应用程序的大小，然后退出。
展示店
显示应用程序和元之间未使用的Flash的大小
数据，然后退出。
显示元
在Bootloader下方显示元数据的大小，然后退出。
展示引导
显示Bootloader的大小，然后退出。
显示版本
显示Bootloader版本和功能，然后退出。
显示向量
显示中断表向量的向量号和名称
Bootloader用于启动应用程序，然后退出。对于
硬件支持的Bootloader，这将是向量 0（重置），并且

对于向量Bootloader，这将是任何其他向量编号
中断向量表或向量表后面的插槽
名称 VBL_ADDITIONAL_VECTOR。

第 2 章：命令行选项
22
展示部分
显示编译Bootloader的部分，然后退出。
靴子尺寸=尺寸
手动覆盖Bootloader大小。 Urboot Bootloader将
将已使用的Bootloader页数放入引导顶部的表中
加载器部分，即通常位于Flash顶部，因此 urclock 编程器
可以查找Bootloader本身的大小。向后兼容
模式，当通过其他Bootloader编程时，此选项可以
用于告诉编程器尺寸，以及位置，
Bootloader的。
向量数=n
手动覆盖矢量编号。 Urboot Bootloader将
向量Bootloader使用的向量编号放入顶部的表中
flash，因此 urboot Bootloader通常不需要此选项。
然而，它在向后兼容模式下很有用（或者当
urboot Bootloader不提供Flash读取）。指定向量
在这些情况下，数字意味着向量Bootloader，同时
默认假设是硬件支持的Bootloader。
乙丙橡胶
用于断言 EEPROM 读/写功能的手动覆盖。不
通常 urboot Bootloader需要，但对于后向有用
如果Bootloader提供 EEPROM 读/写，则为兼容模式。
模拟_ce
如果 urboot Bootloader不提供整片擦除命令，它将
在握手期间告诉 urclock 编程器。在这种情况下
如果用户保证的话，urclock 编程器会模拟整片擦除
命令行选项，通过填充未使用的Flash的剩余部分
使用 0xff 降低Bootloader。如果指定此选项，则 urclock
编程器会假设Bootloader无法擦除芯片
本身。该选项对于向后兼容的Bootloader很有用
不实现整片擦除。
恢复
将未更改的Flash输入文件写入 AVR 并修剪至低于
Bootloader（如果需要）。当有备份时，这是最有用的
完整的Flash并想要将其回放到设备上。否
在这种情况下写入元数据并且不会发生矢量修补
如果它是矢量Bootloader。但是，对于矢量Bootloader，
即使在选项 -x 恢复下，输入文件也不会被写入
到复位向量不指向向量的AVR
Bootloader。这是为了避免将输入文件加载到设备上
这将使矢量Bootloader变得无法访问 af-
重新设置。
初始化存储区
写入Flash时填充Flash应用程序之间的存储空间
和元数据部分为 0xff。

第 2 章：命令行选项
23
无文件名
写入Flash时，不存储应用程序输入文件名（也不存储
一个标题）。
节点
写入Flash时，不存储应用程序输入文件名（也不存储
标题），也没有日期。
诺斯托雷
写入Flash时不存储元数据以外的元数据
代码字节 0xff 表示没有元数据。特别是，没有
数据存储帧已编程。
诺元数据
不支持任何元数据。
除了启动之外的完整Flash
加载程序可用于该应用程序。如果应用较小
大于可用空间，则存储元数据代码字节 0xff
尽管如此，还是要表明没有更多元数据可用。
在缺少 -x nometadata 的情况下，urclock 程序的默认设置 -
mer 是写入尽可能多的元数据（文件名、数据并存储在-
形成）作为应用程序的大小和其他扩展
选项允许。 -x nometadata 和 -x nometadata 之间的细微差别
-x nostore 是后者总是显式地存储在Flash中
没有更多的元数据可用，因此这样准备好的Flash可以
始终使用 avrdude -x showall 进行查询。与此相反，它
无法保证Flash上的 -x showall 查询已准备好
使用 -x nometadata 会产生有用的结果。
不自动重置
不要在端口打开时切换 RTS/DTR 线路，以防止硬件损坏
重置。
延迟=n
复位后添加 n ms 延迟。如果董事会采用
退出外部复位的时间特别长。 n 可以为负数，
在这种情况下，发出重置后默认的 120 毫秒延迟将是
相应缩短。
严格
Urclock 的策略比 -c arduino 更快，但略有不同
与Bootloader同步；一些 stk500v1 Bootloader可以-
无法解决这个问题，他们需要 -x strict 选项。
巴士海盗
BusPirate 编程器类型接受以下扩展参数：
重置=cs,aux,aux2
默认设置假设 BusPirate 的 CS 输出引脚已连接
到 AVR 侧的 RESET 引脚。然而有可能有
通过 SDI、SDO 和 SCK 连接到同一 BP 的多个 AVR
他们所有人的共同点。在这种情况下，一个 AVR 应该具有
其 RESET 连接到 BusPirate 的 CS 引脚，第二个 AVR 的 RESET

连接到 BusPirate 的 AUX 引脚，并且如果您的 BusPirate 有
AUX2 引脚（仅适用于带有固件的 BusPirate 版本 v1a
3.0 或更高版本）使用它来激活第三个 AVR 上的 RESET。

第 2 章：命令行选项
24
将 BusPirate 和 AVR 解耦可能是个好主意
SPI 总线相互之间使用一个三态总线缓冲器。例如
74HC125 或 74HC244 是带有锁存器的一些不错的候选者
由适当的复位引脚（cs、aux 或 aux2）驱动。否则
一个活动电路中的 SPI 流量可能会干扰编程
另一种设计中的AVR。
spifreq=0..7
0
30 kHz（默认）
1
125kHz
2
250kHz
3
1兆赫
4
2兆赫
5
2.6兆赫
6
4兆赫
7
8兆赫
原始频率=0..3
设置 SPI 速度并使用 Bus Pirate 的二进制“原始线”
模式而不是默认的二进制 SPI 模式：
0
5kHz
1
50kHz
2
100 kHz（仅限固件 v4.2+）
3
400 kHz (v4.2+)
“裸线”模式的唯一优点是不同的 SPI
频率可用。这里没有实现分页写入
模式。
引体向上
启用 Bus Pirate 的内置上拉电阻。这些电阻
在不同电压水平下工作时非常有用。
VPU引脚
Bus Pirate 的总线必须连接到外部电压。对于
示例：将VPU引脚连接到+5V引脚或外部电源
供应。
希兹

在 SPI 上启用 Bus Pirate 的 HiZ 模式，使其能够作为
集电极开路并与外部上拉电路接口。如果
外部目标电路没有上拉电阻，总线海盗将
无法发送数据。
ASCII码
即使固件支持 Bin-，也尝试使用 ASCII 模式
模式（二进制模式）。 BinMode 在固件 2.7 和
较新、较旧的固件要么没有 BinMode，要么有 BinMode
是越野车。
ASCII 模式速度较慢，并且使上述重置=，
spifreq= 和 rawfreq= 参数不可用。
请注意
不保证 ASCII 模式适用于较新的固件版本
sions，并且保留只是为了保持与旧版本的兼容性
固件版本。

第 2 章：命令行选项
25
不分页写入
固件版本 5.10 及更高版本支持二进制模式 SPI com-
使整个页面能够写入 AVR Flash的命令
一次，从而显着提高写入速度。如果使用这个
由于某种原因不需要模式，此选项将禁用它。
没有分页阅读
较新的固件版本支持二进制模式 SPI 命令
一些 AVR 扩展命令。使用“批量内存读取”
from Flash”导致读取速度显着提高。如果使用
由于某种原因不需要此模式，此选项将禁用它。
cpu频率=125..4000
这会将 AUX 引脚设置为输出 n kHz 的频率。正在连接
将 AUX 引脚连接到 MCU 的 XTAL1 引脚，您可以提供
它是一个时钟，例如当它需要外部时钟时，因为
熔丝位设置错误。确保CPU频率至少为四
乘以 SPI 频率。
串行接收超时=1...
这会将串行接收超时设置为给定值。超时
每次 avrdude 等待 BusPirate 提示时都会发生。埃斯-
特别是在 ASCII 模式下，这种情况经常发生，因此设置较小的值
值可以大大加快编程速度。默认值为 100
女士。在大多数情况下，使用 10 毫秒可能有效。
微核Bootloader
Micronucleus 编程器类型接受以下扩展参数：
等待=超时
如果设备未连接，请等待设备插入。
可选超时指定连接超时（以秒为单位）。
如果未指定超时，AVRDUDE 将无限期等待，直到
设备已插入。
小小的Bootloader
Teensy 编程器类型接受以下扩展参数：
等待=超时
如果设备未连接，请等待设备插入。
可选超时指定连接超时（以秒为单位）。
如果未指定超时，AVRDUDE 将无限期等待，直到
设备已插入。
接线
接线编程器类型接受以下扩展参数：
贪睡=n
之后
表演
的
港口
打开
阶段，

AVDUDE
会
在继续之前等待/暂停毫秒
协议同步阶段。
如果出现以下情况，则不会执行 DTR/RTS 切换：
贪睡 > 0。

第 2 章：命令行选项
26
延迟=n
重置后添加 n 毫秒延迟。如果董事会
从外部复位退出需要特别长的时间。 n 可以是
负数，在这种情况下，发出复位后默认延迟 100 毫秒
将相应缩短。
PICkit2
与 PICkit2 编程器的连接：
调压器
PICkit2
快速恢复时间
VPP/MCLR (1)
电源电压
VDD 目标 (2) – 如果 AVR 自供电则可选
接地
接地 (3)
SDI
胚胎植入前诊断 (4)
时钟时钟
PDC (5)
SDO
辅助 (6)
PICkit2 编程器类型接受以下扩展参数：
时钟频率=速率
设置 SPI 时钟速率（以 Hz 为单位）（默认值为 100 kHz）。交替地
-B 或 -i 选项可用于设置周期。
超时=usb-事务-超时
设置 USB 读写超时时间（默认为毫秒）
是 1500 毫秒）。
USBASP
USBasp 编程器类型接受以下扩展参数：
节配置
编程器将使用选项“-e”擦除配置部分（芯片
擦除），而不是整个芯片。
仅适用于 TPI 设备
（微小 4/5/9/10/20/40）。
蜜蜂
xbee 编程器类型接受以下扩展参数：
xbeersetpin=1..7
选择连接到 MCU 的 XBee 引脚 DIO1..7
/重置线。编程器需要知道哪个 DIO 引脚
用于重置到Bootloader。默认 (3) 是 DIO3 引脚
（XBee 引脚 17），但某些商业产品使用不同的
XBee 针。
其余两个必需的 XBee 到 MCU 连接不是
可选 - XBee DOUT 引脚（引脚 2）必须连接到

MCU 的 RXD 线和 XBee DIN 引脚（引脚 3）必须连接到 MCU 的 TXD 线。

jtag2updi

序列号

jtag2updi 和serialupdi 编程器类型接受以下扩展参数：

rtsdtr=低,高
强制 RTS/DTR 线路在整个过程中处于低电平或高电平状态
编程会议。一些编程器可能会使用这个信号
指示 UPDI 编程状态，但这严格来说是硬件
具体的。
如果未提供，将使用驱动程序/操作系统默认值。
linuxspi
linuxspi 编程器类型接受以下扩展参数：
禁用_无_cs
确保编程器不使用 SPI NO CS 位
SPI 驱动程序。该参数对于不支持的内核很有用
CS 线在应用程序外部进行管理。
塞尔普罗格
serprog 编程器类型接受以下扩展参数：
CS
设置要在支持的编程器上使用的片选 (CS)。亲-
支持 0x16 serprog 命令的语法可以有更多
比默认 CS (0) 更高。该选项允许选择这些附加
用于对 AVR 进行编程的国家 CSes (1, 2, ...)。

第 2 章：命令行选项
28
2.4 命令行调用示例
AVRDUDE 错误消息、警告和进度报告通常写入 stderr
在 bash 中，可以通过 2>/dev/null 或使用越来越多的 -q 选项来关闭
来镇压他们。命令的终端输出或带输出的 -U 命令的终端输出
名为 - 的文件被写入标准输出。在一些示例中，为了清楚起见，显示了空行
不是由 AVRDUDE 或 shell 打印的。
使用连接的STK500编程器将文件diag.hex写入ATmega128芯片
到默认串口：

⊠
$ avrdude -p m128 -c stk500 -e -U flash:w:diag.hex
从输入文件 diag.hex 读取Flash的 19278 字节
将 19278 字节写入Flash
写作 | ## | 100% 7.60 秒
阅读| ## | 100% 6.83秒
已验证 19278 字节Flash
阿夫杜德完成了。
谢谢。

相同，但处于安静进度报告（静默）模式 -qq：

⊠
$ avrdude -qq -p m128 -c stk500 -e -U flash:w:diag.hex

使用 && 确认静默 AVRDUDE 命令运行正常：

⊠
$ avrdude -qq -p m128 -c stk500 -e -U flash:w:diag.hex && echo OK
好的

将Flash以原始二进制格式保存到名为 c:/diag flash.bin 的文件中：

⊠
$ avrdude -p m128 -c stk500 -U flash:r:"c:/diag flash.bin":r
读取Flash...
阅读| ## | 100% 6.90 秒
将 19278 字节写入输出文件 diag flash.bin
阿夫杜德完成了。
谢谢。

读取熔丝并以不同格式（十六进制、二进制和八进制）打印它们的值：

⊠
$ avrdude -cusbasp -patmega128 -qq -Ulfuse:r:-:h -Uhfuse:r:-:b -Uefuse:r:-:o
0xbf
0b11000110
第0377章

第 2 章：命令行选项
29
使用默认编程器，将文件 diag.hex 写入Flash，将文件 eeprom.hex 写入 EEP-
ROM，并将扩展熔丝字节、高熔丝字节和低熔丝字节分别设置为0xff、0x89和0x2e：

⊠
$ avrdude -p m128 -U flash:w:diag.hex
\
-U eeprom:w:eeprom.hex \
-U efuse:w:0xff:m
\
-U hfuse:w:0x89:m
\
-U lfuse:w:0x2e:m
处理 -U flash:w:diag.hex:i
从输入文件 diag.hex 读取Flash的 19278 字节
将 19278 字节写入Flash
写作 | ## | 100% 7.60 秒
阅读| ## | 100% 6.81 秒
已验证 19278 字节Flash
处理 -U eeprom:w:eeprom.hex:i
从输入文件 eeprom.hex 读取 3328 字节的 eeprom
将 3328 字节写入 eeprom
写作 | ## | 100% 1.20 秒
阅读| ## | 100% 0.70 秒
3328字节eeprom已验证
处理 -U efuse:w:0xff:m
从输入文件 0xff 读取 efuse 的 1 个字节
向 efuse 写入 1 个字节 (0xFF)，写入 1 个字节，验证 1 个字节
处理 -U hfuse:w:0x89:m
从输入文件 0x89 读取 hfuse 的 1 个字节
向 hfuse 写入 1 个字节（0x89），写入 1 个字节，验证 1 个
处理 -U lfuse:w:0x2e:m
从输入文件 0x2e 中读取 lfuse 的 1 个字节
向 lfuse 写入 1 个字节（0x2E），写入 1 个字节，验证 1 个
阿夫杜德完成了。
谢谢。

将数据从stdin（标准输入）写入EEPROM；没有错误输出意味着一切顺利：

⊠
$ echo '敏捷的棕色狐狸' | avrdude -c usbasp -p attiny13 -qq -U eeprom:w:-:r

执行多个以分号分隔的终端模式命令：

⊠
$ echo '写入eeprom 0 "Bonjour";写ee 0x18 0x12345678；转储 eeprom 0 0x20' | \
avrdude-qqcdryrun-patmega328p -t
0000
42 6f 6e 6a 6f 75 72 00

```
ff ff ff ff ff ff ff ff ff ff ff
|你好.........|
0010
ff ff ff ff ff ff ff ff ff ff ff
78 56 34 12 ff ff ff ff
|........xV4.....|
```

第 2 章：命令行选项
30
与使用-T 相同：

⊠
$ avrdude -qqcdryrun -patmega328p \
-T '写入eeprom 0“Bonjour”；写ee 0x18 0x12345678；转储 eeprom 0 0x20'
0000
42 6f 6e 6a 6f 75 72 00
ff ff ff ff ff ff ff ff ff ff ff
|你好.....|
0010
ff ff ff ff ff ff ff ff ff ff ff
78 56 34 12 ff ff ff ff
|.....xV4.....|

读取 EEPROM 并将内容写入 stdout（标准输出）：

⊠
$ avrdude -qq -cusbasp -pattiny13 -Ueeprom:r:-:i
：20000000E2809954686520717569636B2062726F776E20666F78E280990AFFFFFFFFFFFFFD3
:20002000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE0
:00000001FF

相同，但将 stderr（标准错误输出）重定向到 /dev/null 而不是使用 -qq：

⊠
$ avrdude -cusbasp -pattiny13 -Ueeprom:r:-:i 2>/dev/null
：20000000E2809954686520717569636B2062726F776E20666F78E280990AFFFFFFFFFFFFFD3
:20002000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE0
:00000001FF

使用 Avrdude 输出打印Flash中存在的字符串：

⊠
$ avrdude -pattiny13 -qq -U flash:r:-:r |字符串
主菜单
距离：%d 厘米
退出

列出连接到 USB 的所有 JTAG ICE 的序列号；这是通过指定一个来完成的
无效的序列号，并增加详细级别：

⊠
$ avrdude -c jtag2 -p m128 -P USB:xyz -v
使用端口
: USB:xyz
使用编程器
： jtag2fast
编程器波特率
：115200
找到带有 Serno 的 JTAG ICE：00A000001C6B
找到带有 Serno 的 JTAG ICE：00A000001C3A

找到带有 Serno 的 JTAG ICE：00A000001C30
错误：未找到任何（匹配的）USB 设备 usb:xyz (03eb:2103)
错误：无法为编程器 jtag2fast 打开端口 usb:xyz
阿夫杜德完成了。
谢谢。

第 2 章：命令行选项
31
通过USB连接到序列号以1C37结尾的JTAG ICE mkII，输入
交互式终端模式，列出连接部分的所有命令并退出：

⊠
$ avrdude -c jtag2 -p m649 -P USB:1c:37 -t
avrdude> 帮助
有效命令：
转储
：将内存部分显示为十六进制转储
读
: 转储的别名
迪萨斯姆
：拆解内存部分
写
：将数据写入内存；Flash和 EEPROM 被缓存
保存
：将内存段保存到文件中
备份
：备份存储器到文件
恢复：从文件中恢复存储器
验证
：将内存与文件进行比较
冲水
：与设备同步Flash和EEPROM缓存
中止
：中止Flash和EEPROM写入，即重置r/w缓存
擦除
：执行芯片或内存擦除
配置
：更改或显示零件的配置属性
工厂：将部件重置为工厂状态
regfile：I/O寄存器地址和内容
include：包含指定文件的内容，就像键入该文件一样
信号
：显示设备签名字节
部分
：显示当前零件信息
发送
：发送原始命令给编程器
verbose：显示或设置 -v 详细级别
镇压
：显示或设置 -q 进度条的平息级别
帮助
：显示帮助信息
?
: 与帮助相同

退出
： 将 flash/EEPROM 缓存与设备同步并退出
q
: 退出的缩写
有关终端命令 cmd 类型 cmd -? 的更多详细信息
其他：
！ 线
： 在子 shell 中运行 shell 行，例如 !ls *.hex
...
： 忽略该行的其余部分（例如，用作脚本中的注释）
请注意，并非所有编程器衍生产品都支持所有命令。闪光灯和
EEPROM 类型存储器通常使用高速缓存通过分页进行读写
读写访问；缓存在退出或刷新命令时同步。
part 命令显示用于转储和写入的有效内存。
avrdude> 退出
阿夫杜德完成了。
谢谢。

设备的出厂熔丝位设置：

```
$ avrdude -patmega328p/St | $ avrdude -patmega328p/St | grep 初始值
.ptmm ATmega328P lfuse 初始值 0x62
.ptmm ATmega328P hfuse 初始值 0xd9
.ptmm ATmega328P efuse 初始值 0xff
.ptmm ATmega328P 锁定初始值 0xff
```

AVRDUDE 已知的所有部件列表：

```
$ avrdude -p \*/d |切-f2-d"'"
AT小11
ATtiny12
AT小13
ATtiny13A
AT小15
AT89S51
[...]
AVR64EA48
LGT8F88P
LGT8F168P
LGT8F328P
```

AVRDUDE 已知的所有现代 AVR 部件（带有 UPDI 接口）的列表：

```
$ avrdude -p \*/Ud |切-f2-d"'"
ATtiny202
AT小204
AT小402
[...]
AVR64EA28
AVR64EA32
AVR64EA48
```

libserialport 库已知的所有当前插入的串行设备的列表：

```
$ avrdude -P \?s
可能的候选串行端口是：
-P /dev/ttyUSB0 或 -P ft232r:A600K203
-P /dev/ttyUSB1 或 -P ft232r:MCU8
-P /dev/ttyUSB3、-P ch340 或 -P ch340-115k
请注意，上述端口可能未连接到目标板或 AVR 编程器。
另请注意，可能还有上面未列出的其他直接串行端口。
```

第 2 章：命令行选项
33
AVRDUDE 已知的所有串口适配器的列表，即在 avrdude.conf 中定义的：

☒
$ avrdude -P \?sa
有效的串口适配器有：
第340章
= [usbvid 0x1a86，usbpid 0x7523]
ch341a
= [usbvid 0x1a86，usbpid 0x5512]
第342章
= [usbvid 0x1a86，usbpid 0x55d2]
第343章
= [usbvid 0x1a86，usbpid 0x55d3]
第344章
= [usbvid 0x1a86，usbpid 0x55d5]
第347章
= [usbvid 0x1a86，usbpid 0x55da 0x55db 0x55dd 0x55de]
ch9102
= [usbvid 0x1a86，usbpid 0x55d4]
ch9103
= [usbvid 0x1a86，usbpid 0x55d7]
CP210X
= [usbvid 0x10c4，usbpid 0xea60 0xea70 0xea71]
ft2232h = [usbvid 0x0403，usbpid 0x6010]
ft231x
= [usbvid 0x0403，usbpid 0x6015]
ft234x
= [usbvid 0x0403，usbpid 0x6015]
ft230x
= [usbvid 0x0403，usbpid 0x6015]
英尺232小时
= [usbvid 0x0403，usbpid 0x6014]
ft232r
= [usbvid 0x0403，usbpid 0x6001]
ft4232h = [usbvid 0x0403，usbpid 0x6011]
PL2303
= [usbvid 0x067b，usbpid 0x2303 0x2304 0x23a3 0x23b3 0x23c3 0x23d3 0x23e3]

输出可用于某个部件的非Bootloader编程器的列表。注意 2>&1
在 bash shell 中将 stderr 折叠到 stdout 中：

☒
$ avrdude -c“？” -p 32ea32 2>&1 | -p 32ea32 2>&1 | grep -v Bootloader
AVR32EA32 部件的有效编程器有：
atmelice_updi
= Atmel-ICE (UPDI)
试运行

= 无需编程器即可模拟编程（TPI，[...]）
jtag2updi
= JTAGv2 至 UPDI 桥 (UPDI)
纳米每个
= JTAGv2 至 UPDI 桥 (UPDI)
jtag3updi
= Atmel AVR JTAGICE3 (UPDI)
pickit4_mplab_updi
= MPLAB(R) PICkit 4 (UPDI)
pickit4_updi
= MPLAB(R) PICkit 4 (UPDI)
pickit5_updi
= MPLAB(R) PICkit 5 (UPDI)
pickit_basic_updi
= MPLAB(R) PICkit 基本型 (UPDI)
pickit_basic_mplab_updi = MPLAB(R) PICkit Basic (UPDI)
pkobn_updi
= 好奇号纳米 (nEDBG) (UPDI)
powerdebugger_updi
= Atmel 电源调试器 (UPDI)
序列号
= 串行UPDI (UPDI)
snap_mplab_updi
= MPLAB(R) SNAP (UPDI)
快照updi
= MPLAB(R) SNAP (UPDI)
xplainedmini_updi
= Atmel XplainedMini (UPDI)
xplainedpro_updi
= Atmel XplainedPro (UPDI)

第 2 章：命令行选项
34
打印 AVRDUDE 理解的编程器定义：

⊠
```
$avrdude -c linuxspi/s
#----------------------------------------------------------------------
#linuxspi
#----------------------------------------------------------------------
# 该编程器使用内置的linux SPI总线设备来编程
# 附加 AVR。复位引脚必须连接到 GPIO 引脚
# 未使用（请参阅 gpioinfo(1)）； SPI总线CE引脚不是
# 合适，因为他们会过早地释放/RESET。
#
编程器#linuxspi
编号
=“linuxspi”；
描述
= "在 /dev/spidev* 中使用 Linux SPI 设备";
类型
=“linuxspi”；
编程模式
= PM_TPI | PM_ISP；
额外功能
= HAS_BITCLOCK_ADJ；
连接类型
= SPI；
重置
= 25；
# Pi GPIO 编号 - 这是 J8:22
;
```

打印最后存储的草图的文件名及其日期戳（仅适用于 urclock 编程器）：

⊠
```
$avrdude -qq -curclock -P/dev/ttyUSB0 -pattiny13 -x showdate -x showfilename
2023-05-19 11.13 闪烁.hex
```

bash 脚本中的 AVRDUDE 创建将部件重置为出厂设置的终端脚本：

⊠
```
$ cat make-init-脚本
#!/bin/bash
mkdir /tmp/工厂
对于 $(avrdude -p\*/d | grep = | cut -f2 -d"'") 中的部分；做
回显$部分
avrdude -p$部分/St | grep 初始值 |切 -f3,5 | grep -ve-1 \
| sed "s/.*/write &/" >/tmp/factory/$part.ini
完成
```

运行上面的脚本并使用创建的终端脚本之一：

⊠

```
$ ./make-init-scripts
$猫/tmp/factory/ATmega328P.ini
写入 lfuse 0x62
写入hfuse 0xd9
写入efuse 0xff
写锁0xff
$ avrdude -qq -cusbasp -pATmega328P -t < /tmp/factory/ATmega328P.ini
```

第 2 章：命令行选项
35
创建一个 bash 函数 avrdude-elf，它将 elf 文件作为输入，支持可选
Avrdude 在末尾标记，并写入 elf 文件中指定的所有内存。在这个例子中，
elf 文件不包含任何 EEPROM 数据：

⊠

```
# 显示 ATtiny13 存在的所有可写内存
$ echo $(avrdude -pattiny13/ot | grep write | cut -f3 | uniq)
eeprom Flash lfuse hfuse 锁
# 写入 elf 文件中所有内存的函数
avrdude-elf() {
avrdude -cusbasp -pattiny13 -U{eeprom,flash,{l,h}熔丝位,锁}:w:"$1":e "${@:2}"
}
# 运行函数，其中 -B8 和 -V 附加到 Avrdude 命令后
$ avrdude-elf眨眼.elf-B8-V
将 SCK 频率设置为 93750 Hz
处理 -U eeprom:w:blink.elf:e
从输入文件blink.elf中读取eeprom的64字节
将 64 字节写入 eeprom
写作 | ################################################## | 100% 0.08秒
写入 64 字节 eeprom
处理 -U flash:w:blink.elf:e
从输入文件blink.elf中读取flash的1024字节
将 1024 字节写入Flash
写作 | ################################################## | 100% 0.12秒
已写入 1024 字节Flash
处理 -U lfuse:w:blink.elf:e
从输入文件blink.elf中读取lfuse的1个字节
向lfuse写入1个字节（0x6A），写入1个字节
处理 -U hfuse:w:blink.elf:e
从输入文件blink.elf中读取hfuse的1个字节
向hfuse写入1个字节（0xFF），写入1个字节
处理 -U lock:w:blink.elf:e
从输入文件blink.elf中读取1个字节以进行锁定
写入1字节（0xFF）锁定，写入1字节
阿夫杜德完成了。
谢谢。
```

36
3 终端模式操作
AVRDUDE 有一种称为终端模式的交互模式，通过 -t 选项启用。
此模式允许输入交互式命令来显示和修改各种 de-
副存储器，执行整片擦除，显示设备签名字节和部分参数，
并发送原始编程命令。命令和参数可以缩写
以其最短的明确形式。终端模式还支持命令历史记录，因此
可以调用和编辑以前输入的命令。
3.1 终端模式命令
在此模式下，AVRDUDE仅初始化与MCU的通信，然后等待
标准输入上的用户命令。命令和参数可以缩写为
最短的明确形式。终端模式使用 readline(3) 提供命令历史记录，
因此可以调用和编辑以前输入的命令行。
dump、read、disasm、write、save 和 Erase 命令的 addr 和 len 参数
可以为负数，其语法与 perl 或 python 中的子字符串计算相同。的
下表定义了给定内存大小 sz 的有效内存间隔 [start, end]：
地址
伦
存储器间隔
评论
0/正
积极的
[地址，地址+len-1]
注：len = 结束-开始 + 1
0/正
消极的
[地址，sz+len]
结束是|len|低于 mem 大小 sz 的字节
消极的
积极的
[sz+地址，sz+地址+len-1]
起始地址为 |addr|字节低于内存大小
消极的
消极的
[sz+地址，sz+长度]
结合以上两种情况
任何
零
空集
无动作
请注意，addr 必须在 [-sz, sz-1] 范围内。
计算出内存区间后
[start,end]如上表所示，有效长度end-start + 1不能为负数；如果
有效长度为零则不执行任何操作。结束可能超出可用范围
用于转储、读取或 disasm 命令的内存，在这种情况下操作会回绕
内存端，但有效长度始终受限于内存大小。
以下是 sz 大小为 0x800 (2048) 字节的内存的一些示例：

地址
伦
存储器间隔
评论
0x700
12
[0x700，0x70b]
常规使用
1024
-257
[0x400，0x6ff]
0x6ff = 2048-257
-512
第512章
[0x600，0x7ff]
最后 512 字节
-256
-1
[0x700，0x7ff]
最后 256 字节
0
49
[0, 48]
前 49 个字节
0
-49
[0, 1999]
除最后 48 之外的所有 = |len+1|字节
0
-1
[0, 0x7ff]
所有内存，不知道其大小
2046
4
[0x7fe，0x801]
读取时环绕但写入时出错
2046
4096
[0x7fe，0x17fe]
读取环绕停止于 0x7fd
-1
-1
[0x7ff，0x7ff]
0x7ff 处的一个字节被寻址
-1
-2
[0x7ff，0x7fe]
无动作：有效长度为零

-1
-3
[0x7ff，0x7fd]
错误：有效长度为负数

第 3 章：终端模式操作
37
以下命令适用于所有编程器：
转储内存地址长度
从指定的内存区间读取（见上文），并以通常的方式显示
十六进制和 ASCII 形式。
转储内存地址
从内存地址读取与最近转储内存地址一样多的字节
已指定此内存的 len 命令（默认 256 字节），并且
显示它们。
转储内存
Continue dumping from the memory and location where the most recent dump
命令停止；如果之前没有转储命令已寻址内存
将显示错误消息。
转储
从内存和最近转储的位置继续转储
命令停止；如果之前没有转储命令已寻址内存
将显示错误消息。
转储内存地址...
从 addr 开始读取，一直到最后一个内存地址（已弃用：
使用转储内存地址-1)。
转储内存...
从指定内存中读取所有字节，并显示它们（不推荐使用：使用
转储内存 0 -1)。
读
可以用作 dump 的别名。
disasm [选项] 转储参数
与 dump 一样，disasm 命令显示指定内存的一部分，al-
通过将内存内容解释为 AVR 操作码并将其显示为
汇编源代码。与 dump 不同，disasm 命令有选项；这些
控制 disasm 如何显示其结果（见下文）。除此之外，语法
指定内存及其要处理的时间间隔实际上是相同的
与 dump 一样：默认的 disasm 长度是 32 字节，但有时
长度可以比要求的稍短或稍长，以便内存
disasm 部分与操作码一致。解除选项一旦设置，就保持有效
直到关闭，通常是通过更改选项的大小写。这样，一个
无需进一步选项的简单 disasm 可用于单步执行内存
保持外观。 Disasm 知道以下选项：
-g
生成 avr-gcc 源：设置 -sOFQ 并输出 .text pream-
ble 和一个主符号，除非反汇编本身发出一个符号； -G
（默认）关闭 -g 并停止输出前导码
-A
不显示地址； -a（默认值）显示地址
-O
不显示操作码字节； -o（默认）显示操作码字节
-C

不显示评论； -c（默认）显示注释
-f
显示 SREG 中受影响的标志，例如，sbiw 操作码的 ---SVNZC；
-F（默认值）不显示 SREG 标志

第 3 章：终端模式操作
38
-q
显示操作码所花费的机器周期数； -Q（
默认）不显示周期
-n
将操作码全名放入注释中（例如，减去立即数
来自单词）； -N（默认值）不显示完整的操作码名称
-e
将操作码的技术解释放入注释中，例如，
Rd+1:Rd <-- Rd+1:Rd - K 为 sbiw 操作码； -E（默认值）做
不显示技术说明
-S
使用 AVR 指令集风格：这意味着寄存器对是
例如，显示为 r31:30 而不是 r30； -s（默认）使用avr-gcc
代码风格
-L
不对标签进行预处理； -l（默认）预处理跳转/调用
标签
-U
不要展示未使用的标签； -u（默认值）显示未使用的标记
标签
-d
解码所有操作码，包括那些未记录的操作码； -D（
默认）仅解码对该部分有效的操作码
-z
Zap 反汇编前的跳转和调用列表
-t=文件
从先前读取的标记文件中删除符号（如果有），然后读取
用于将地址分配给符号名称的标记文件文件。
标记文件是一个 ASCII 文件，其中每一行描述一个代码标签符号
地址 (L)、Flash中的变量地址 (P) 和内存中的变量地址
或 I/O 空间 (M)。哈希标记开始延伸到末尾的标记文件注释
该行并被 disasm 忽略。这是标记文件如何定义的示例
看起来像

⊠
0x7f54 L
普奇
输出一个 char # L 是代码标签
0x7ffe 密码 1
version16 A 字整数 # P 是 PGM 数据
0x7f80 帕4
标题
列标题 # 自动对齐字符串
0x0100 MB 2048 随机存储器
2kB 静态存储器
内存地址

代码标签L可以是例如程序空间中的函数名称或goto标签。他们
最多使用由空格分隔的四列：地址、字母 L、
标签的符号名称和符号的可选注释列，
由 disasm 复制到反汇编注释栏中，应该这样
代码引用或使用标签。变量符号中带有 P 或 M
第二栏；它们可以是字节或字（16 位），由字母决定
第三列中的 B 或 W 以及指定的单个变量或数组
第四列中的多重数。 P 符号可以，但 M 符号不行
还可以编码字符（8 位）、长整型（32 位）、四元组（64 位）或八元组（128 位）
分别由字母 C、L、Q 或 O 表示，或者是基准位置
由第三列中的 A 或 S 编码的以 null 结尾的字符串。出
必要时，A/S字符串占用的空间各不相同。
之间的区别
A 和 S 符号是 A 字符串数组可能有一个额外的 nul

第 3 章：终端模式操作
39
字符将它们占用的空间自动对齐到偶数地址。的
第五列是可以使用的 P 或 M 地址的符号名称
通过 disasm 以符号方式输出相关地址。 P 区域中描述
tagfile还告诉disasm对应的区域不是代码，不应该
被如此拆卸；相反，指令用于反汇编该指令
区。与 L 标签一样，P 和 M 变量可能有可选的最终注释
与反汇编列中可能输出的符号相关的列
as 和 当使用相应的变量时。
标记文件对于反汇编很有用，可以使 disasm 的输出更具可读性。
它们可以根据人们的理解手动和增量地构建
代码增长。
或者，bash shell 脚本 elf2tag 可以自动
从生成 Flash 内容的 .elf 文件生成标记文件：

```
$ elf2tag application.elf >application.tag
```

elf2tag 使用 avr-objdump -d 反汇编来创建 L 标签和 avr-nm
生成M个符号。
写入内存地址 data[,] {data[,]}
从地址 addr 开始，使用以下命令对各个存储单元进行手动编程
提供的数据项。终端实现读写
通常通过一个Flash、EEPROM、bootrow 和 usersig 类型存储器
缓存和分页访问功能。所有其他存储器都直接写入
不使用缓存。一些较旧的部分没有分页访问，具体取决于
编程器也可能直接访问Flash和 EEPROM，而无需
缓存。项目数据可以具有以下格式：
类型
示例
大小（字节）
字符串
“你好，世界\n”
变化的
文件
C:/My\projects/blink.hex
变化的
带格式的文件
闪烁.hex:i
变化的
性格
'A'
1
二进制整数
0b101010
1、2、4 或 8
八进制整数
012345

1、2、4 或 8
十进制整数
12345
1、2、4 或 8
十六进制整数
0x12345
1、2、4 或 8
十进制浮点数
3.1415926
4
十六进制浮点数
0xA.8p2
4
十进制双精度
3.141592653589793D
8
十六进制双精度
0xA.8p2D
8
数据可以是二进制、八进制、十进制或十六进制整数、浮点数
bers 或 C 风格的字符串和字符。如果没有任何匹配，数据将被交互
解释为包含数据的文件的名称，该文件将被读取并插入到
这一点。为了强制将数据项解释为文件，例如，当
文件名将被理解为数字，否则文件名可以是
给定 :f 格式说明符。如果没有格式后缀，终端将尝试
自动检测文件格式。

第 3 章：终端模式操作
40
以 urboot 开头的文件名：自动生成只读的 urboot boot-
加载程序文件。例如尝试终端命令 write flash urboot:help
了解决定Bootloader内容的功能列表。参见章节-
有关详细文档，请参阅第 5 [自动生成的文件]，第 62 页。值得
这里注意，在终端中将 urboot:... 文件写入Flash并不会写入
Bootloader工作所需的熔丝位（与 -U 操作相反）。
对于整数，可选的不区分大小写的后缀指定数据大小：
LL
8 字节（64 位）
L
4 字节（32 位）
H或S
2字节（16位）
赫赫
1 字节（8 位）
后缀 D 表示 64 位双精度型，F 表示 32 位浮点型，而浮点数
不带后缀的默认为 32 位浮点数。十六进制浮点表示法是
支持。不明确的尾部后缀，例如 0x1.8D，会被读取为无后缀浮点数
其中 D 是尾数的一部分；使用零指数 0x1.8p0D 来澄清。
可选的 U 后缀使整数无符号。普通0x十六进制和0b
二进制整数始终被视为无符号。 +0x、-0x、+0b 和 -0b 数字
带有显式符号的符号将被视为有符号符号，除非它们具有 U 后缀。未签名
整数不能大于 2^64-1。如果 n 是无符号整数，那么 -n 也是
与 C 中一样的有效无符号整数。有符号整数必须落入 [-2^63, 2^63-
1] 范围或当后缀指定较小的范围时相应较小的范围
类型。
普通 0x 十六进制和 0b 二进制整数，有 n 个十六进制数字（计数
前导零）使用一、二、四和八字节的最小大小
容纳任何 n 位十六进制/二进制整数。如果整数后缀指定
明确指定大小，相应的最低有效字节数是
写入，如果该数字不适合所需的表示，则会显示警告
感觉。否则，无符号整数占据 1、2、4 或 中最小的一个
需要八个字节。有符号数可以放入最小的有符号数中
或最小的无符号表示形式：例如，255 作为一个字节存储为
255U 适合一个字节，但作为有符号数，它不适合一个字节
一字节间隔 [-128, 127]。数字-1存储在一个字节中，而-1U
需要八个字节，因为它与 0xFFFFffffFFFFffffU 相同。
数据项末尾的一个逗号将被忽略，以方便复制和复制
列表的粘贴。
写入内存数据
如果要写入的存储器的大小是，则可以省略起始地址
一个字节。数据可以是任何东西，包括文件。
写入内存文件
当文件写入内存时，起始地址可以被省略。
写入内存 addr len data[,] {data[,]} ...

省略号。。。 form 将数据写入整个内存区间
通过 addr len，如有必要，通过重复最后一个来填充剩余空间
数据项。填充写入命令不会写入超出指定内存的内容
即使提供的数据多于所需的数据。

第 3 章：终端模式操作
41
保存内存 {addr len} 文件[:格式]
以 :format 指定的格式将一个或多个内存段保存到文件中
信。默认为 :r 表示原始二进制文件。每个内存段都有描述
通过地址和长度对。在没有任何内存段的情况下，整个
内存被保存到文件中。仅 Motorola S-Record (:s) 和 Intel Hex (:i
或：I) 格式将地址信息与保存的数据一起存储。阿夫杜德不能
当前保存 ELF 文件格式。
所有其他文件格式都会丢失地址
信息并将所选内存段连接到输出文件中。
如果文件名是 - 则 avrdude 写入 stdout。
备份memlist文件[:格式]
使用所选格式将一个或多个存储器备份到指定文件。
单内存备份的默认格式是：r（原始二进制文件）；对于多
内存备份是：I（带有注释的英特尔十六进制）。 memlist 可以是逗号
分隔的内存列表就像 -U 命令行参数中一样。备份
在读取内存之前刷新缓存。
恢复memlist文件[:格式]
从指定文件恢复一个或多个存储器。这是用户的责任
能够根据需要事先擦除内存：一些分页内存看起来像
当使用某些编程器时，NOR-内存，这意味着编程不能
将位设置为 1（例如，大多数编程器下的Flash）。
这些存储器需要
使用擦除命令预先擦除（见下文）。仅格式
如果无法自动检测，则需要指定，例如，当文件被
- 用于标准输入。 Memlist 可以是逗号分隔的内存列表
如 -U 命令行参数中所示。恢复在写入之前刷新缓存
写入内存并在写入内存后重置缓存。注意恢复
只读存储器用相应的微处理器验证文件内容
索尔的回忆。
验证 memlist 文件[:格式]
将一个或多个内存与指定文件进行比较。 memlist 可以是逗号
分隔的内存列表就像 -U 命令行参数中一样。验证
在验证内存之前刷新缓存。
擦除
执行整片擦除并丢弃所有待写入Flash、EEPROM 和
引导。
请注意，如果 EESAVE 熔丝位为 EEPROM 将被保留
活动的，即在操作之前的最后一次重置时具有相应的值。
擦除存储器
擦除整个指定内存。
擦除内存地址长度
擦除指定内存的一部分。
冲水
与器件同步所有待写入Flash、EEPROM、引导行和
用户名。通过一些编程器和部件组合，Flash（有时

EEPROM 也）看起来像 NOR 存储器，即写入只能清除位，而不能清除
设置它们。对于 NOR 存储器，需要进行页面擦除，或者如果不可用，则需要进行整片擦除
在写入任意数据之前发出。 Usersig 不受整片擦除的影响。
当存储器看起来像 NOR 存储器时，会部署任一页擦除（例如，

具有 PDI/UPDI 接口的部件），或者如果不可用，则两者
EEPROM和Flash缓存已完全读入，发出整片擦除命令
并且 EEPROM 和Flash都被写回设备。因此，它可以
需要几分钟的时间来确保设置一个先前清除的位，因此，
应谨慎调用此例程。
中止
通常，缓存仅在使用刷新时才实际写入设备，
在输入 quit 后或在输入 EOF 之后的终端会话结束时
遇到的。
abort 命令重置缓存，丢弃所有先前的内容
写入Flash、EEPROM、bootrow 和 usersig 缓存。
配置[-f|-a|-v]
显示零件的所有配置属性；这些通常是熔丝中的位域
或锁定可以采用值的位字节，这些值通常具有助记符
名字。每个部分都有自己的一组可配置项。选项 -f 组
由它们所在的熔丝和锁定位字节决定的配置属性，
并显示这些存储器的当前价值。 config -a 输出一个
具有所有属性和所有可能的相应分配的初始化脚本。
当前分配的助记词值是未注释的值
出来。选项 -v 增加 config 命令输出的详细程度。
配置 [-f|-v] 属性 [-f|-v]
显示指定配置属性的当前值。
通配符或
允许初始字符串（但不能同时使用两者），在这种情况下，当前值
显示所有匹配的属性。
配置 [-f|-v] 属性= [-f|-v]
显示指定配置属性的所有可能值（注意尾随
=）。当前设置的一项是唯一未注释掉的一项。和以前一样，
允许使用通配符或初始字符串。
配置 [-f|-v|-c] 属性=值 [-f|-v|-c]
将指定的配置属性修改为给定值。相应的
熔丝位或锁定位将立即更改，但更改通常只会
下次重置该部件时生效，此时熔丝位和锁
位被利用。值可以是有效整数或符号之一
助记符（如果知道）。允许使用通配符或初始字符串
属性或指定的助记符值，但仅在以下情况下才会进行赋值
属性和名称都可以唯一解析。选项 -v 显示
通过再次读取分配的配置属性的值
熔丝位。如果没有 -v，选项 -c 会确认配置的新值
仅当属性已更改时。
这是很有可能的，就像直接写入底层熔丝和锁一样
位，使零件变砖，即使其对进一步编程无响应
被选中的编程器：这里是龙。
恢复出厂设置
尽可能将连接的部分重置为出厂状态（Bootloader，例如

例如，无法写入熔丝并且可能无法擦除 EEPROM）。这个命令可能会更改下一个命令之后部分的时钟频率 F CPU

第 3 章：终端模式操作
43
当更改的熔丝值生效时，MCU 复位。因此，这可能
要求未来的 avrdude 调用使用不同的位时钟速率，最高可达 F CPU/4
下次给编程器。请注意，命令工厂可以缩写为
viated 但所需的参数重置需要完整拼写。
regfile [选项]
没有进一步参数的 regfile 显示一部分的寄存器文件，即所有
如果可能的话，在 io 内存中注册名称及其内容：注意外部
编程无法读取经典部分（ISP或TPI接口）的寄存器。
选项-a另外显示寄存器I/O地址； -m 显示reg-
用于 lds/sts 操作码的内存地址而不是 I/O 地址。
选项 -s 还显示寄存器的大小（以字节为单位），而 -v 显示稍微
每个寄存器旁边都有扩展的寄存器解释。
regfile [选项] reg [选项]
regfile 与寄存器名称 reg 一起显示所有这些寄存器
由 reg 匹配。允许使用通配符或部分字符串，但不能同时使用两者。注册-
名称的形式为 module.name 或 module.instance.name。如果亲
vided reg 是一个完整的、现有的寄存器名称，例如 porta.out 那么这是唯一的
即使它可能是另一个名称的部分名称，也会显示该寄存器
注册，例如 porta.outdir。如果提供的reg与instance.name相同或
名称，则不再使用部分匹配，并且所有模块都注册为
显示确切的实例名称或名称。可以强制部分匹配
通过使用通配符，例如使用 porta.out*
regfile [选项] reg=值 [选项]
这会将 reg 寻址的单个寄存器设置为给定值。仅外部
现代零件（具有 UPDI 接口的零件）的编程可以读取和
写入寄存器 io 内存，但由于该内存是易失性的，内容将
重置后会丢失。
包含 [opts] 文件
包含指定文件 file 的内容，就像键入该文件一样。这对于
批处理脚本，例如熔丝位的重复初始化代码。包含选项 -e
在处理文件之前将文件的行打印为注释；在一个非零的
详细级别也会打印行号。
签名
显示设备签名字节。
部分[选择]
显示当前零件信息，包括支持的编程模式，
内存和变量表。使用 -m 仅打印内存表，使用 -v 则
只打印变体表。
详细[级别]
更改（当提供级别时）或显示详细级别。
最初的
详细级别由命令上给出的 -v 选项的数量控制
线。

第 3 章：终端模式操作
44
镇压[等级]
更改（当提供级别时），或显示压制级别。 1 用于抑制
进度报告。 2 或更高的产量逐渐变得更安静的操作。最初的
抑制级别由命令行上给出的 -q 选项数量控制。
?
帮助
给出可用命令的简短在线摘要。
退出
离开终端模式，从而离开 AVRDUDE。
q
可以用作 quit 的别名。
！线
在子 shell 中运行 shell 行，例如 !ls *.hex。
子shell命令
将剩下的队伍作为他们的命令。
出于安全原因，他们
必须通过设置allow_subshells = yes;显式启用进入你的
${HOME}/.config/avrdude/avrdude.rc 或 ${HOME}/.avrduderc 文件。
comment 将注释放在终端行上（对脚本有用）。
此外，某些编程器还支持以下命令：
pgerase内存地址
擦除指定内存的一页。
发送 b1 b2 b3 b4
将原始指令代码发送到 AVR 设备。如果您需要访问某项功能
AVRDUDE 不直接支持的 AVR 部分，此命令
允许您使用它，即使 AVRDUDE 没有实现该命令。
使用直接 SPI 模式时，最多可以省略 3 个字节。
SPI
进入直接 SPI 模式。 pgmled 引脚用作片选。仅支持
并行 bitbang 编程器，部分由 USBtiny 提供。片选必须是
使用 USBtinyISP 时，外部将直接 SPI 保持为低电平，并且发送必须是
四字节的倍数。
铂族金属
返回编程模式（从直接 SPI 模式）。
目标电压
将目标的电源电压设置为电压伏特。
varef [通道]电压
将可调电压源设置为电压伏特。该电压通常为
用于驱动 STK500 和 STK600 上目标的 Aref 输入。 STK600
提供两个参考电压，可通过可选参数选择
通道（0 或 1）。
fosc 频率[M|k]
将编程振荡器设置为频率 Hz。可选的尾随字母 M 多
层由 1E6 组成，尾随字母 k 由 1E3 组成。
焦点关闭

关闭编程振荡器。
sck周期
将 SCK 时钟周期设置为周期微秒。
请注意，一些官方
Microchip 编程器存储位时钟设置并将继续使用它
直到提供不同的值。有关详细信息，请参阅 -B 位时钟。

第 3 章：终端模式操作
45
帕尔姆斯
显示编程器特定参数。

第 3 章：终端模式操作
46
3.2 终端模式示例
进入终端，显示零件参数，修改EEPROM，执行整片擦除并退出：

⊠
$ avrdude -qq -c usbasp -p atmega328p -t
avrdude> 部分
ATmega328P 具有编程模式 ISP、HVPP、debugWIRE、SPM
内存
尺寸
PG尺寸

EEPROM
1024
4
闪光
32768
128
流出物
1
1
熔断器
1
1
熔断器
1
1
锁
1
1
签名
3
1
校准
1
1
io
224
1
斯拉姆
2048
1
变体
套餐
最大频率
温度范围
电压范围

ATmega328P
不适用
20兆赫
[-40°C，
不适用]
[1.8V、5.5V]
ATmega328P-15MZ
MLF32
20兆赫
[-40°C，
85°C]
[1.8V、5.5V]
ATmega328P-AN
TQFP32
20兆赫
[-40°C、105°C]
[1.8V、5.5V]
ATmega328P-ANR
TQFP32
20兆赫
[-40°C、105°C]
[1.8V、5.5V]
ATmega328P-AU
TQFP32
20兆赫
[-40°C，
85°C]
[1.8V、5.5V]
ATmega328P-AUR
TQFP32
20兆赫
[-40°C，
85°C]
[1.8V、5.5V]
ATmega328P-MMH
MLF28
20兆赫
[-40°C，
85°C]
[1.8V、5.5V]
ATmega328P-MMHR
MLF28
20兆赫
[-40°C，
85°C]
[1.8V、5.5V]
ATmega328P-MN

QFN32
20兆赫
[-40°C、105°C]
[1.8V、5.5V]
ATmega328P-MNR
MLF32
20兆赫
[-40°C、105°C]
[1.8V、5.5V]
ATmega328P-MU
MLF32
20兆赫
[-40°C，
85°C]
[1.8V、5.5V]
ATmega328P-MUR
MLF32
20兆赫
[-40°C，
85°C]
[1.8V、5.5V]
ATmega328P-PN
PDIP28
20兆赫
[-40°C、105°C]
[1.8V、5.5V]
ATmega328P-PU
PDIP28
20兆赫
[-40°C，
85°C]
[1.8V、5.5V]
avrdude> 转储 eeprom 0 16
0000
ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff
|.....|
avrdude> 写入 eeprom 0 1 2 3 4 0xcafe "Avrdude"
avrdude> 转储 eeprom 0 16
0000
01 02 03 04 铁 ca 41 76
72 64 75 64 65 00 FF FF
|……阿夫杜德……|
avrdude> 冲洗
avrdude> 擦除
擦除芯片...
avrdude> 转储 eeprom 0 16
0000

```
ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff
|................|
avrdude> 退出
```

第 3 章：终端模式操作
47
使用熔丝计算器对 ATmega328P 的熔丝位进行编程
- 启用具有长启动时间的全摆幅高速外部晶振
- 删除默认时钟分频 8
- 重置跳转至Bootloader
- 将Bootloader的大小设置为 512 字节（256 字）
- 在 2.7 V 时启用欠压检测

首先显示出厂默认值，然后查阅外部熔丝位计算器，选择AT-mega328P部分，找到上面的设置，记下三个熔丝位的后续新值和重新编程：

```
$ avrdude -c usbasp -p atmega328p -t
avrdude> 转储 efuse
阅读| ################################################## | 100% 0.00 秒
0000
FF
|。
|
avrdude> 转储 hfuse
阅读| ################################################## | 100% 0.00 秒
0000
d9
|。
|
avrdude> 转储 lfuse
阅读| ################################################## | 100% 0.00 秒
0000
62
|b
|
avrdude>#
avrdude> # 查阅外部熔丝位计算器
avrdude>#
avrdude>#
avrdude> 写入 efuse 0xfd
写作 | ################################################## | 100% 0.01秒
avrdude> 写入 hfuse 0xde
写作 | ################################################## | 100% 0.01秒
avrdude> 写入 lfuse 0xf7
写作 | ################################################## | 100% 0.01秒
avrdude> 退出
阿夫杜德完成了。
谢谢。
```

第 3 章：终端模式操作
48
使用 config 命令对 ATmega328P 的熔丝位进行编程

⊠
$ avrdude -qq -c usbasp -p atmega328p -t
avrdude> # 显示所有配置
avrdude>配置
配置 sut_cksel=intrcosc_8mhz_6ck_14ck_65ms # 34
配置 ckout=co_disabled # 1
配置 ckdiv8=by_8 # 0
配置 bootrst=应用程序#1
配置 bootsz=bs_2048w # 0
配置 eesave=ee_erased # 1
配置 wdton=wdt_programmable # 1
配置 spien=isp_enabled # 0
配置 dwen=dw_off # 1
配置 rstdisbl=external_reset # 1
配置 bodlevel=bod_disabled # 7
配置 lb=no_lock # 3
配置 blb0=no_lock_in_app # 3
配置 blb1=no_lock_in_boot # 3
avrdude> # 显示全摆幅外部晶振的可能值
avrdude> 配置 sut_cksel=extfs
avrdude 警告：（配置）不明确；已知的 sut_cksel extfs 符号有：
- sut_cksel=extfsxtal_258ck_14ck_4ms1 #
6
- sut_cksel=extfsxtal_1kck_14ck_65ms
#
7
- sut_cksel=extfsxtal_258ck_14ck_65ms # 22
- sut_cksel=extfsxtal_16kck_14ck_0ms
#23
- sut_cksel=extfsxtal_1kck_14ck_0ms
#38
- sut_cksel=extfsxtal_16kck_14ck_4ms1 # 39
- sut_cksel=extfsxtal_1kck_14ck_4ms1
#54
- sut_cksel=extfsxtal_16kck_14ck_65ms # 55
avrdude> # 设置适当的启动时间
avrdude> c 苏=55
avrdude> # 取消时钟除以 8 的编程，使复位跳转到Bootloader
avrdude> c ckdiv8=1
avrdude> c bootrst=启动
avrdude> c bootsz=bs_256w
avrdude> # 查询存在哪些bod级别；将欠压设置为 2.7 V
avrdude> c bodlevel=
#conf bodlevel=bod_4v3

#4
#conf bodlevel=bod_2v7
#5
#conf bodlevel=bod_1v8
#6
配置 bodlevel=bod_disabled #7（工厂）
avrdude> c bod=*2v7
avrdude> 退出

第 3 章：终端模式操作
49
显示和更改寄存器

⊠
$ avrdude -c xplainedmini_updi -p ATtiny817 \
-T“reg ctrlc”-T“reg usart0.baud=0x1234”-T“reg -asv usart0”
处理-T reg ctrlc
0x00 端口复用.ctrlc
0x00 adc0.ctrlc
0x03 usart0.ctrlc
0x00 tca0.ctrlc
0x00 tcd0.ctrlc
处理-T reg usart0.baud=0x1234
处理-T reg -asv usart0
输入/输出 0x800：(1)
0x00 usart0.rxdatal
接收数据低字节
输入/输出 0x801：(1)
0x00 usart0.rxdatah
接收数据高字节
输入/输出 0x802：(1)
0x00 usart0.txdatal
发送数据低字节
输入/输出 0x803：(1)
0x00 usart0.txdatah
发送数据高字节
输入/输出 0x804：(1)
0x20 usart0.状态
状态寄存器
输入/输出 0x805：(1)
0x00 usart0.ctrla
控制寄存器A
输入/输出 0x806：(1)
0x00 usart0.ctrlb
控制寄存器B
输入/输出 0x807：(1)
0x03 usart0.ctrlc
控制寄存器C
I/O 0x808：(2) 0x1234 usart0.baud
波特率寄存器（16位）
输入/输出 0x80b：(1)
0x00 usart0.dbgctrl
调试控制寄存器
输入/输出 0x80c：(1)
0x00 usart0.evctrl
事件控制寄存器
输入/输出 0x80d：(1)

0x00 usart0.txplctrl #IRCOM发送器脉冲长度控制寄存器
输入/输出 0x80e：(1)
0x00 usart0.rxplctrl #IRCOM接收器脉冲长度控制寄存器
阿夫杜德完成了。
谢谢。

显示经典部分的寄存器文件

☒
$ avrdude -qq -p ATtiny11 -c dryrun -T“regfile -av”
I/O 0x08：ac.acsr
模拟比较器控制和状态寄存器
I/O 0x16：端口b.pinb
B端口输入寄存器
I/O 0x17：端口b.ddrb
B端口数据方向寄存器
I/O 0x18: portb.portb # 端口 B 数据寄存器
I/O 0x21：wdt.wdtcr
看门狗定时器控制寄存器
I/O 0x32：tc0.tcnt0
定时器/计数器0
I/O 0x33：tc0.tccr0
T/C 0 控制寄存器
I/O 0x34：cpu.mcusr
单片机状态寄存器
I/O 0x35：cpu.mcucr
单片机控制寄存器
I/O 0x38：tc0.tifr
T/C中断标志寄存器
I/O 0x39：tc0.timsk
T/C中断屏蔽寄存器
I/O 0x3a：exint.gifr
通用中断标志寄存器
I/O 0x3b: exint.gimsk # 通用中断屏蔽寄存器
I/O 0x3f：cpu.sreg
状态寄存器

第 3 章：终端模式操作
50
下面的例子演示了负地址和长度字节，以及填充形式
使用省略号的写入命令...其中提供的最后一个数据项用于填充
向上指定的内存范围。

☒

```
$ avrdude -c usbasp -p atmega328p -t
avrdude>转储Flash-64 -33
阅读| ################################################## | 100% 0.02秒
7fc0
ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff
|................|
7fd0
ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff
|................|
avrdude>转储Flash-32 -1
阅读| ################################################## | 100% 0.00 秒
7fe0
ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff
|................|
7ff0
ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff
|................|
avrdude> 写入 flash -64 1234567890 'A' 'V' 'R' 2.718282 0xaa 0xbb 0xcc “你好世界！”
缓存 | ################################################## | 100% 0.00 秒
avrdude> 写入Flash -32 -1 0x01 0b00000010 0b11 0x04 0x05 ...
缓存 | ################################################## | 100% 0.00 秒
avrdude> 读取Flash -64 -33
阅读| ################################################## | 100% 0.00 秒
7fc0
d2 02 96 49 41 56 52 55
f8 2d 40 aa bb cc 48 65
|...IAVRU.-@...他|
7fd0
6c 6c 6f 20 57 6f 72 6c
64 21 00 FF FF FF FF FF
|世界！.....|
avrdude>读取Flash-32 -1
阅读| ################################################## | 100% 0.00 秒
7fe0
01 02 03 04 05 05 05 05
05 05 05 05 05 05 05 05
|................|
```

7ff0
05 05 05 05 05 05 05 05
05 05 05 05 05 05 05 05
|.....|
avrdude> 冲洗
正在将缓存同步到设备...
写作 | ## | 100% 0.05秒
avrdude> 退出
阿夫杜德完成了。
谢谢。

第 3 章：终端模式操作
51
反汇编ATtiny13A的Flash内容，将输出写入文件blink.S，编译
到 ‘blink.elf’ 并验证 ATtiny13A 的 flash 内容是否与 ATtiny13A 的 flash 内容相同
由编译后的blink.elf给出。然后将反汇编的blink.S猫到标准输出。

⊠
$ avrdude -qq -p t13a -c avrisp2 -T "disasm -g flash 0 -1" >blink.S
$ avr-gcc -mmcu=attiny13a -nostdlib -Wl,--section-start=.text=0x0000 闪烁.S -o bl.elf
$ avrdude -qq -p t13a -c avrisp2 -U flash:v:bl.elf && echo OK
好的
$ 猫眨眼.S
.equ
io.pinb，0x16
.equ
io.ddrb，0x17
.text
主要：
L000:
rjmp
标签1
; L016
L002：
rjmp
标签0
; L014
L004：
rjmp
标签0
; L014
L006：
rjmp
标签0
; L014
L008：
rjmp
标签0
; L014
L00a：
rjmp
标签0
; L014
L00c：
rjmp
标签0
; L014
L00e：
rjmp

标签0
; L014
L010：
rjmp
标签0
; L014
L012：
rjmp
.+0
; L014
;来自 L002、L004、L006、L008、L00a、L00c、L00e、L010 的 Rjmp
标签0：
L014：
雷蒂
标签1：
;来自 L000 的 Rjmp
L016：
履行机构
io.ddrb, 2
;位 2 = 0x04
标签2：
;来自 L024 的 Rjmp
L018：
LDI
r29，0x1e
; 30
标签3：
; Brne 来自 L01c、L020
L01a:
斯比威
r30, 0x01
; 1
L01c：
布尔内
标签3
; L01a
L01e：
十二月
r29
L020：
布尔内
标签3
; L01a
L022：
履行机构
io.pinb, 2
;位 2 = 0x04
L024：

rjmp
标签2
; L018
L026：
.填充
493, 2, 0xffff

第 3 章：终端模式操作
52
混合终端命令和 -U 内存操作：下面的示例烧录了一个引导-
加载器，使用终端线将应用程序数据写入Flash，加载应用程序，配置
将掉电检测电平设置为 2.7 V，最后将完整Flash存储为新的十六进制文件。
请注意 bash 中使用不同的引号来传递终端命令行：
AVRDUDE 的单个实体。

```
$ avrdude -qc dryrun -p m328p \
-U urboot_m328p_1s_autobaud_uart0_pr_ee_ce.hex \
-T '写入 flash 0x7D00 0xc0cac01a 0xcafe “可口可乐秘密配方”' \
-U flash:w:cola-vending-machine.hex \
-T “配置-v bod=*2v7” \
-U flash:r:app+data.hex:I
处理 -U flash:w:urboot_m328p_1s_autobaud_uart0_pr_ee_ce.hex:i
从输入文件 urboot_m328p_1s_autobaud_uart0_pr_ee_ce.hex 读取Flash的 368 字节
将 368 字节写入Flash，写入 368 字节，验证 368 字节
处理-T write flash 0x7D00 0xc0cac01a 0xcafe“可口可乐秘密配方”
正在将缓存同步到设备...完成
处理 -U flash:w:cola-vending-machine.hex:i
从输入文件 cola-vending-machine.hex 读取 736 字节的Flash
将 736 字节写入Flash，写入 736 字节，验证 736 字节
处理 -T config -v bod=*2v7
配置 bodlevel=bod_2v7 # 5
处理 -U flash:r:app+data.hex:I
读取Flash...
将 32768 字节写入输出文件 app+data.hex
阿夫杜德完成了。
谢谢。
```

53

4 配置文件

AVRDUDE 在启动时读取一个配置文件，该文件描述了所有部件和
它所了解的编程器。这样做的好处是，如果你有一个芯片
AVRDUDE 目前不支持，您可以将其添加到配置文件中，无需
等待 AVRDUDE 的新版本。同样，如果您有并行端口编程器
不支持，您可能可以复制现有的编程器定义
只需进行一些更改，即可让您的编程器正常工作。
AVRDUDE 首先在依赖于平台的系统中查找系统范围的配置文件
位置。在 Unix 上，这通常是 /usr/local/etc/avrdude.conf 请参阅第 A.1 节 [Unix]，
第 73 页，而在 Windows 上，它通常与可执行文件位于同一位置。的
可以使用 -C 命令行选项指定该文件的全名。解析后
系统范围的配置文件，AVRDUDE 会查找每个用户的配置文件来
增强或覆盖系统范围的默认值。在 Unix 上，每用户文件是 ${XDG_CONFIG_
HOME}/avrdude/avrdude.rc，而如果 ${XDG_CONFIG_HOME} 未设置或为空，
使用 ${HOME}/.config/ 代替。如果用户的目录中不存在 .avrduderc
使用主目录。在 Windows 上，此文件是位于同一目录中的 avrdude.rc 文件
目录作为可执行文件。

4.1 AVRDUDE 默认值

avrdude_conf_version = "构建时版本";
在构建过程中自动设置。
default_parallel = "默认并行设备";
分配默认并行端口设备。可以使用 -P 选项覆盖。
default_serial = "默认串行设备";
分配默认串行端口设备。可以使用 -P 选项覆盖。
default_linuxgpio = “默认linuxgpio设备”;
为 linuxgpio 的 libgpiod 模式分配默认的 gpiochip，例如“GPIO芯片0”。
linuxgpio 的 sysfs 模式被忽略。可以使用 -P 选项覆盖。
default_programmer = "默认编程器 ID";
分配默认编程器 ID。可以使用 -c 选项覆盖。
default_baudrate = "默认波特率";
如果编程器不这样做，则指定将使用的默认波特率值
提供其具体的波特率条目。可以使用 -b 选项覆盖。
default_bitclock = "默认位时钟";
分配默认位时钟值。可以使用 -B 选项覆盖。
允许子壳=否；
AVRDUDE的交互终端是否允许使用subshell
！ 命令。出于安全原因，这默认为 no，例如，在极少数情况下
avrdude -t 与附加硬件一起设置以提供远程 Web 服务
ssh 或在 PC 上登录而不是 shell，例如用于演示或培训目的。在
几乎所有其他情况都可以在每用户 avrddude.rc 中覆盖或
.avrduderc 配置文件带有 yes。

第 4 章：配置文件
54
4.2 编程器定义
编程器定义的格式如下：
编程器
父级<id>
可选的父级
编号
= <id1> [, <id2> ...];
<idN> 是带引号的字符串
描述
= <描述>;
带引号的字符串
类型
= <类型>；
# 编程器类型，带引号的字符串
# 使用 -c ?type 列出已知类型
prog_modes = PM_<i/f> {|下午_<i/f>}
# 接口，例如 PM_SPM|PM_PDI (1)
is_serialadapter = <是|否>
# 编程器也是一个串口适配器
extra_features = HAS_<fea> {| HAS_<fea>}
# 额外的功能，例如 HAS_SUFFER (2)
连接类型=并行|连续剧 | USB | SPI
波特率 = <数字>;
avr910-programmer 的波特率
电压控制电路
= <引脚1> [, <引脚2> ...];
引脚编号 (3)
浅黄色
= <引脚1> [, <引脚2> ...];
引脚号
重置
= <引脚>；
引脚号
sck
= <引脚>；
引脚号
sdo|pico = <引脚>;
引脚号
sdi|poci = <pin>;
引脚号
tck
= <引脚>；
引脚号
tdi
= <引脚>；

引脚号
tdo
= <引脚>;
引脚号
TMS
= <引脚>;
引脚号
犯了错误
= <引脚>;
引脚号
雷德莱德
= <引脚>;
引脚号
PGLED
= <引脚>;
引脚号
维勒德
= <引脚>;
引脚号
USB视频
= <十六进制数>;
USB 供应商 ID
usbpid
= <十六进制数> [, <十六进制数> ...];
USB 产品 ID (4)
USB设备
= <接口>;
# USB接口或其他设备信息
usbvendor = <供应商名称>;
# USB 供应商名称
usbproduct = <产品名称>;
# USB 产品名称
USB网络
= <序列号>;
# USB 序列号
hvupdi_support = <num> [, <num>, ...];
# HV 支持启用 UPDI
;
如果指定了父级，则它的所有设置（除了它的 id）都将用于新编程器。
这些值可以通过为新编程器进行新设置来更改。
注释
1. 已知的编程模式有
- PM_SPM：Bootloader，使用 SPM 操作码或 NVM 控制器进行自编程
- PM_TPI：微型编程接口（t4、t5、t9、t10、t20、t40、t102、t104）
- PM_ISP：用于系统内编程的SPI编程（几乎所有经典部分）
- PM_PDI：编程和调试接口（xmega 部件）
- PM_UPDI：统一编程和调试接口
- PM_HVSP：高压串行编程（一些经典部分）

- PM_HVPP：高压并行编程（大多数非HVSP 经典部件）
- PM_debugWIRE：JTAG 的更简单替代方案（HVPP/HVSP 部件的子集）
- PM_JTAG：联合测试行动组标准（一些经典部分）
- PM_JTAGmkI：PM_JTAG 的子集、旧部件、Atmel ICE mkI
- PM_XMEGAJTAG：JTAG，一些 XMEGA 部件

第 4 章：配置文件
55
・ PM_AVR32JTAG：用于 32 位 AVR 的 JTAG
・ PM_aWire：AVR32 部件
2. 以下额外的编程器功能是已知的
・ HAS_SUFFER：仅出现在 Xplained Mini/Nano 编程器上；超级用户
神奇功能启用寄存器允许用户修改
mEDBG 编程器/调试器芯片，请参阅 Xplained Mini/Nano 文档
欲了解更多信息
・ HAS_VTARG_SWITCH：编程器有一个可编程目标电源开关
・ HAS_VTARG_READ：编程器可以读取目标电压
・ HAS_VTARG_ADJ：编程器有一个可调节的目标电源，可以
由 Avrdude 控制
・ HAS_FOSC_ADJ：编程器有一个可编程频率发生器，可以提供时钟
直接通过其 XTAL1 引脚连接 AVR
・ HAS_VAREF_ADJ：编程器具有可调节的模拟参考电压，可以
由 Avrdude 控制
3. 要反转引脚的极性，请使用波形符 ~<num>；反转所有引脚的极性
列表使用 ~(<num1> [, <num2> ...])
4. 并非所有编程器类型都可以处理 USB PID 列表
当前实现了以下编程器类型：
阿尔杜伊诺
用于引导加载的 Arduino 编程器
AVR910
使用应用笔记 AVR910 中的协议的串行编程器
阿夫夫迪
使用 libftdi 与 FTDI 芯片的 MPSSE 引擎接口
avrftdi_jtag
libftdi JTAG 接口
丁螺酯
Bus Pirate 的 SPI 接口
巴士海盗_bb
Bus Pirate 的 bitbang 界面
蝴蝶
Atmel Butterfly 评估板（AVR109、AVR911）
蝴蝶_mk
蝴蝶微型飞行器
ch341a
芯片 CH341A：AVR 的 CPU 最低频率必须为 6.8 MHz
试运行
用于测试 avrdude 的 Dryrun 编程器
龙_dw
Atmel AVR Dragon 处于 debugWire 模式
龙_hvsp
Atmel AVR Dragon 处于 HVSP 模式
龙ISP
Atmel AVR Dragon 处于 ISP 模式

龙_jtag
JTAG 模式下的 Atmel AVR Dragon
龙_pdi
Atmel AVR Dragon 处于 PDI 模式
龙_pp
Atmel AVR Dragon 处于 PP 模式
翻转1
FLIP USB DFU 协议版本 1 (doc7618)
翻转2
FLIP USB DFU 协议版本 2 (AVR4023)
ftdi_syncbb
FT245R/FT232R 同步 bit-bang 编程器
杰塔姆基
Atmel JTAG ICE mkI
杰塔姆基伊
Atmel JTAG ICE mkII
jtagmkii_avr32
AVR32 模式下的 Atmel JTAG ICE mkII
jtagmkii_dw
debugWire 模式下的 Atmel JTAG ICE mkII
jtagmkii_isp
ISP 模式下的 Atmel JTAG ICE mkII

第 4 章：配置文件
56
jtagmkii_pdi
PDI 模式下的 Atmel JTAG ICE mkII
jtagmkii_updi
UPDI 模式下的 Atmel JTAG ICE mkII
jtagice3
爱特梅尔JTAGICE3
jtagice3_pdi
PDI 模式下的 Atmel JTAGICE3
jtagice3_updi
Atmel JTAGICE3 处于 UPDI 模式
jtagice3_dw
Atmel JTAGICE3 处于 debugWire 模式
jtagice3_isp
Atmel JTAGICE3 处于 ISP 模式
jtagice3_tpi
TPI 模式下的 Atmel JTAGICE3
linuxgpio
使用 Linux libgpiod 或 sysfs 接口进行 GPIO bitbanging
linuxspi
使用Linux spidev驱动程序的SPI
微核
微核Bootloader
标杆
并行端口位碰撞
挑选2
Microchip 的 PICkit2 编程器
挑选5
Microchip 的 PICkit 5 编程器/调试器
塞族人
串行端口位碰撞
序列号
SerialUPDI 编程器的驱动程序
塞尔普罗格
通过 Flashrom 的 Serprog 协议进行编程
STK500
Atmel STK500 1.x 版固件
STK500通用
Atmel STK500，自动检测固件版本
STK500V2
Atmel STK500 2.x 版固件
stk500hvsp
Atmel STK500 V2 处于高压串行编程模式
STK500PP
Atmel STK500 V2 并行编程模式
STK600

爱特梅尔STK600
STK600HVSP
Atmel STK600 处于高压串行编程模式
STK600PP
Atmel STK600 并行编程模式
很小的
青少年Bootloader
你的时钟
用于 urboot Bootloader的 Urclock 编程器（兼容 arduino）
usbasp
USBasp编程器，参见http://www.fischl.de/usbasp/
微型USB
Usbtiny 编程器（也作为引导加载协议）
接线
使用 STK500v2 协议的Bootloader (AVR068)
蜜蜂
XBee 系列 2 无线 (XBeeBoot)
4.3 串口适配器定义
串口适配器定义的格式如下：
串口适配器
父级<id>
可选的串口适配器或编程器父级
编号
= <id1> [, <id2> ...];
<idN> 是带引号的字符串
描述
= <描述>;
带引号的字符串
波特率 = <数字>;
可选的默认波特率，例如，在 .avrduderc 中
USB视频
= <十六进制数>;
USB 供应商 ID
usbpid
= <十六进制数> [, <十六进制数> ...];
USB 产品 ID 列表
USB网络
= <序列号>;
每个用户 .avrduderc 中的 USB 序列号
;
从技术上讲，串口适配器是作为只有 USB 参数的编程器来实现的
已定义。它可用于 -P <serialadapter>[:<serial number>] 端口指定
阳离子而不是创建的串行端口。 ~/.avrduderc 中的每用户串口适配器定义
或 avrdude.rc 文件可以添加序列号来为特定板分配特定 id 和
默认通讯波特率：

第 4 章：配置文件
57
串口适配器父级“ft232r”
编号
=“自行车棚门”；
USB网络
=“0123456789”；
波特率
= 250000；
;
这对于通过Bootloader进行编程特别有用，因为它允许指定
端口为 -P bike-shed-door 而不必弄清楚哪个串行端口名称
操作系统已在运行时分配给插入的自行车棚门板。注意事项
每个定义 usbpid 并设置 is_serialadapter = yes 的编程器也可以
用作串口适配器。
4.4 零件定义
部分
描述
= <描述>;
带引号的字符串，长部件名称，例如“ATmega328p”
编号
= <id>；
带引号的字符串，通常是缩写的部件名称
变体
= <str1> [, <str2> ...]; # 带引号的字符串，每个字符串都以“<alt-name>: ...”开头
家庭 ID
= <id>；
带引号的字符串，例如“megaAVR”或“tinyAVR”
编程模式
= PM_<i/f> {|下午_<i/f>}
接口，例如 PM_SPM|PM_ISP|PM_HVPP|PM_debugWIRE
麦克伊德
= <数字>;
8 位 AVR 的 0..2039 中的唯一 ID
阿克努姆
= <数字>;
avr-gcc架构编号；如果不是 8 位 AVR，则为 -1
n_中断
= <数字>;
中断数量，用于向量Bootloader
n_page_erase
= <数字>;
如果设置，则在 SPM 擦除期间擦除的页数
n_boot_sections
= <数字>;
# 引导部分的数量
boot_section_size = <数字>;

#（最小）引导部分的大小（如果有）
hvupdi_variant
= <数字>;
数字：-1 或 0..3；如何通过 HV 启用 UPDI
stk500_开发代码
= <数字>;
数字
avr910_开发代码
= <数字>;
数字
is_at90s1200
= <是/否>;
AT90S1200部分
签名
= <数字> <数字> <数字>;
签名字节
usbpid
= <数字>;
#DFU USB PID
chip_erase_delay = <数字>;
# 微秒
重置
= 专注 | io；
重试脉冲
=重置| sck；
STK500参数（并行编程IO线）
帕格尔
= <数字>;
十六进制页面加载引脚名称，例如 0xD7
BS2
= <数字>;
字节选择 2 十六进制引脚名称，例如 0xA0
连续剧
= <是/否>;
可以使用串口编程
平行
= <是/否/伪>;
可以使用并行编程
STK500v2 参数，取自 Atmel 的 ATDF 文件
超时
= <数字>;
稳定延迟
= <数字>;
cmd执行延迟
= <数字>;
同步环路
= <数字>;
字节延迟

= <数字>;
民意调查值
= <数字>;
民意调查指数
= <数字>;
预延迟
= <数字>;
延迟后
= <数字>;
轮询方法
= <数字>;
hvspcmdexedelay
= <数字>;
STK500v2 HV 编程参数，来自 ATDF
pp_controlstack
= <数字>, <数字>, ...;
# 仅限PP
hvsp_controlstack = <编号>, <编号>, ...;
# 仅 HVSP
Flash指令
= <数字>, <数字>, <数字>;
EEPROM_指令
= <数字>, <数字>, ...;
hventerstabdelay = <数字>;

第 4 章：配置文件
58
程序模式延迟
= <数字>;
仅限PP
锁存周期
= <数字>;
切换vtg
= <数字>;
关机延迟
= <数字>;
重置延迟
= <数字>;
重置延迟
= <数字>;
hvleavestabdelay = <数字>;
复位延迟
= <数字>;
同步周期
= <数字>;
# 仅 HVSP
Chiperasepulsewidth = <num>；
# 仅限PP
Chiperasepolltimeout = <num>；
芯片时间
= <数字>;
仅 HVSP
程序熔丝位脉冲宽度 = <num>;
仅限PP
程序fusepolltimeout = <num>;
程序锁脉冲宽度 = <num>;
仅限PP
程序锁轮询超时 = <num>;
# debugWIRE 和/或 JTAG ICE mkII 参数，也来自 ATDF 文件
allowedfullpagebitstream = <是/否>;
启用页面编程 = <是/否>;
印尼盾
= <数字>;
IDR (OCD) 寄存器的 IO 地址
拉姆兹
= <数字>;
RAMPZ 寄存器的 IO 地址
精微分析仪
= <数字>;
SPMC[S]R 寄存器的内存地址
埃克
= <数字>;

EECR 寄存器的内存地址
埃因德
= <数字>;
EIND 寄存器的内存地址
单片机库
= <数字>;
ATxmega 设备中的 MCU 控制块
nvm_base
= <数字>;
ATxmega 设备中的 NVM 控制器
ocd_base
= <数字>;
AVR8X/UPDI 设备中的 OCD 模块
系统配置库
= <数字>;
AVR8X/UPDI 设备中的芯片版本 ID
奥德雷夫
= <数字>;
ATDF 文件中的 JTAGICE3 参数
pgm_启用
=<指令格式>;
整片擦除
=<指令格式>;
Bootloader的参数
自动波特率同步
= <数字>;
自动波特率检测字节，默认0x30
工厂_fcpu
= <数字>;
F_CPU（以 Hz 为单位）重置和出厂设置的熔丝位
内存 <memstr>
寻呼的
= <是/否>;
是/否（仅闪现经典部分）
偏移量
= <数字>;
内存偏移量
尺寸
= <数字>;
字节
页面大小
= <数字>;
字节
页数
= <数字>;
数字
初始值
= <数字>;

熔丝位和锁位的出厂设置
位掩码
= <数字>;
使用的位（仅在熔丝和锁定位中）
n_word_写入
= <数字>;
# 仅 TPI：如果设置，则要写入的字数
min_write_delay = <数字>;
# 微秒
max_write_delay = <数字>;
# 微秒
回读
= <数字> <数字>;
字节值对
回读_p1
= <数字>;
字节值（第一个组成部分）
回读_p2
= <数字>;
# 字节值（第二个部分）
pwroff_after_write = <是/否>;
# 是/否
模式
= <数字>;
ATDF 文件中的 STK500 v2 文件参数
延迟
= <数字>;
#
”
块大小
= <数字>;
#
”
读取大小
= <数字>;
#
”
读
=<指令格式>;
写
=<指令格式>;
读洛
=<指令格式>;
读_hi
=<指令格式>;

第 4 章：配置文件
59
写洛
=<指令格式>;
写_hi
=<指令格式>;
加载页面_lo
=<指令格式>;
加载页面_hi
=<指令格式>;
写页
=<指令格式>;
;
;
如果未指定上述任何参数，则使用默认值 0 作为数字
（除了 mcuid、hvupdi_variant、ocdrev、initval 和 bitmask，所有这些都默认为
-1，对于 autobaud_sync，默认为 0x30）或字符串值的空字符串“”。
如果必需的参数留空，AVRDUDE 会抱怨。几乎所有出现的情况
数字（引脚编号以及它们之间用空格分隔的情况除外，例如，
在签名和回读中）也可以作为涉及算术和的简单表达式给出
按位运算符。
4.4.1 父部分
部件还可以使用以下语法从先前定义的部件继承参数。
在这种情况下，指定的整数和字符串值会覆盖来自父级的参数值
部分。新的内存定义将添加到从父级继承的定义中。如果，
然而，新的内存定义引用了该部分现有的同名内存定义
然后，从v7.1开始，扩展了现有的内存定义，并覆盖了组件
具有新的价值观。分配 NULL 会删除继承的 SPI 指令格式、内存
定义、控制堆栈、eeprom 或 flash 指令，例如内存中的“efuse”= NULL;。
变量参数永远不会被继承，因为这样做几乎总是错误的：
变体定义了一个字符串列表，详细说明了零件的变体名称，后跟可选的
冒号、封装代码和一些绝对最大额定值。
部分继承的示例格式：
父部分 <id>
标识父级的字符串
编号
= <id>；
新零件的 ID 字符串
<上面列表中的任何其他参数集>
;
4.4.2 指令格式
指令格式被指定为逗号分隔的字符串值列表，其中包含
有关指令 32 位中每一位的信息（位说明符）。位说明符可以
可以是以下格式之一：
1
该位始终在输入和输出上设置
0

该位在输入和输出时始终清零
x
该位在输入和输出时被忽略
一个
该位是地址位，位号与该位说明符的位置相匹配
在当前指令字节内
一个N
该位为第 N 个地址位，位号 = N，即 a12 为地址位 12
输入，a0 是地址位 0。
我
该位为输入数据位；与位一样，输入数据位可以选择
后跟一个位编号，这里在 0 到 7 之间，如果该位需要移动到
SPI 写入命令字节中的位置与内存中出现的位置不同。

第 4 章：配置文件
60
哦
该位为输出数据位；与 i 位一样，输出数据位可以选择
后面跟着一个位数；这在部件的 SPI 读取命令的情况下很有用
将特定位放置到与写入命令放置的位置不同的位置，
例如，ATtiny22L 或 AT90S8535 锁定位。
每条指令必须由 32 位说明符组成。指令规范
严格遵循 Atmel 零件数据表中提供的说明数据。对于
例如，AT90S2313 AVR 部件的 EEPROM 读写指令可以是
编码为：
读
=“1
0
1
0
0
0
0
0
x x x x
x x x x”，
“x a6 a5 a4
a3 a2 a1 a0
噢噢噢
哦哦哦”；
写=“1
1
0
0
0
0
0
0
x x x x
x x x x”，
“x a6 a5 a4
a3 a2 a1 a0
我我我我
我我我我”；
由于 SPI 操作码中的地址位数是高度系统化的，因此它们并不真正
需要指定。格式规范的紧凑版本既不使用位数
用于地址行或空格。如果这样的字符串长度超过 7 个字符，则字符
0、1、x、a、i 和 o 将被识别为相应的位，而任何字符
.、-、_ 或 / 可以充当任意视觉分隔符，但会被忽略。示例：
loadpage_lo = "0100.0000--000x.xxxx--xxaa.aaaa--iiii.iiii";
loadpage_lo = "0100.0000", "000x.xxxx", "xxaa.aaaa", "iiii.iiii";

4.5 其他注意事项

・ stk500_devcode 参数为STK500使用的设备代码，获取
来自 Atmel AVR061 应用说明的软件部分 (avr061.zip)
来自http://www.atmel.com/dyn/resources/prod_documents/doc2525.pdf。

・ 并非所有存储器都会执行所有指令。

・ AVR 熔丝位和锁定位作为一种存储器来实现。

・ 示例存储器有：Flash、eeprom、熔丝、lfuse（低熔丝）、hfuse（高熔丝）、
efuse（扩展熔丝位）、签名、校准、锁定。

・ AVRDUDE 命令行上指定的内存必须与以下内存之一匹配：
为指定芯片定义的ory。

・ pwroff_after_write 标志导致 AVRDUDE 尝试关闭设备电源
如果 VCC 编程器对受影响的内存区域进行了不成功的写入，则返回
引脚已定义。如果没有为编程器定义 VCC 引脚，则会出现一条消息，指示
打印出设备需要重新启动。添加此标志是为了解决此问题
at90s4433/2333 的问题；请参阅 at90s4433 勘误表：
https://www.microchip.com/content/dam/mchp/documents/OTH/
产品文档/数据表/doc1042.pdf

・ 应用笔记 AVR109 中的Bootloader（因此也是 AVR Butterfly）
不支持熔丝位写入。支持写入锁定位，但仅限于
引导锁定位 (BLBxx)。这些是底层 SPM 施加的限制

用于从Bootloader内部对设备进行编程的指令。请注意
对启动锁定位进行编程可能会导致“搬进你的脚”的情况
取消对这些位的编程的唯一方法是整片擦除，这也会擦除Bootloader
代码。
Bootloader通过擦除Flash页来实现“整片擦除”功能
应用部分。
完全支持读取熔丝和锁定位。

62
5 自动生成的文件
自动生成的文件是虚拟的只读文件，外部不存在，但
由 AVRDUDE 提供。它们具有形式前缀：参数并且可以使用
写入内存的内容。前缀告诉 AVRDUDE 要生成哪种类型的文件
而冒号后面的部分通常是下划线分隔的参数列表，
确定自动生成的文件的内容。
目前，唯一自动生成的文件是来自 https:// 的 urboot Bootloader
github.com/stefanrueger/urboot 项目。
它们采用 urboot:features 的形式。
Bootloader功能使用下划线分隔的列表以任意顺序指定，例如，
urboot：autobaud_2s。有关可能功能的完整列表，请参阅下文。以下
示例将合适的自动波特率Bootloader写入 ATmega328P 目标的Flash：

```
$ avrdude -c dryrun -p m328p -U urboot:autobaud
从输入文件 urboot:autobaud 读取Flash的 260 字节
将 260 字节写入Flash
写作 | ################################################## | 100% 0.00 秒
阅读| ################################################## | 100% 0.00 秒
260 字节Flash已验证
设置Bootloader urboot:autobaud 的熔丝位
阿夫杜德完成了。
谢谢。
```

自动生成的Bootloader也可以在终端中使用：

```
$ avrdude -c dryrun -p m328p -t
avrdude> 写入Flash urboot:autobaud
缓存 | ################################################## | 100% 0.00 秒
avrdude> 退出
正在将缓存同步到设备...
写作 | ################################################## | 100% 0.00 秒
阿夫杜德完成了。
谢谢。
```

请注意，在 -U 选项中使用 urboot:... 具有同时设置所有内容的预期副作用
Bootloader正常工作所需的熔丝位配置。这不会发生
使用终端写命令时，需要进行必要的熔丝配置
手动请求。

第 5 章：自动生成的文件
63
终端模式非常适合交互式探索可能的Bootloader：

⊠
$ avrdude -c dryrun -p m328p -t
avrdude> 写入Flash urboot:autobaud_list
选择尺寸用途与功能
类型
规范文件名
_pr *256 256 u8.0 w---jPra-向量/spmready
urboot_m328p_1s_autobaud_u...
_ee_ce_pr *378 384 u8.0 weU-jPrac 矢量/spmready
urboot_m328p_1s_autobaud_u...
_ee_u4_pr
372 384 u8.0 weU-jPra-矢量/spmready
urboot_m328p_1s_autobaud_u...
_ce_u4_pr
362 384 u8.0 w-U-jPrac 向量/spmready
urboot_m328p_1s_autobaud_u...
_u4_pr
316 384 u8.0 w-U-jPra-矢量/spmready
urboot_m328p_1s_autobaud_u...
_ee_ce_u4_hw *400 512 u8.0 weU-hprac 硬件支持的 urboot_m328p_1s_autobaud_u...
_ee_ce_u4_pr
414 512 u8.0 weU-jPrac 向量/spmready
urboot_m328p_1s_autobaud_u...
_ee_u4_hw
354 512 u8.0 weU-hpra-硬件支持的urboot_m328p_1s_autobaud_u...

列表功能不会生成Bootloader代码，而是以交互方式列出
可能的选择。在上面的示例中，由于空间原因，最后一列被截断。使用
依次为 ee（提供具有 EEPROM r/w 功能的Bootloader）、ce（添加整片擦除
生成的Bootloader的仿真代码）和u4（生成代码，跳过所有冗余
Flash页写入和页擦除可减少Flash的磨损并加快
引导加载过程）然后将迭代地减少可用的选择：

⊠
avrdude> 写入Flash urboot:autobaud_list_ee
选择尺寸用途与功能
类型
规范文件名
_ce_pr *378 384 u8.0 weU-jPrac 矢量/spmready
urboot_m328p_1s_autobaud_uart...
_u4_pr
372 384 u8.0 weU-jPra-矢量/spmready
urboot_m328p_1s_autobaud_uart...
_ce_u4_hw *400 512 u8.0 weU-hprac 硬件支持的 urboot_m328p_1s_autobaud_uart...
_ce_u4_pr

414 512 u8.0 weU-jPrac 向量/spmready
urboot_m328p_1s_autobaud_uart...
_u4_hw
354 512 u8.0 weU-hpra-硬件支持的urboot_m328p_1s_autobaud_uart...
avrdude> 写入Flash urboot:autobaud_list_ee_ce
选择尺寸、用途与功能
类型
规范文件名
_pr *378 384 u8.0 weU-jPrac 矢量/spmready
urboot_m328p_1s_autobaud_uart0_r...
_u4_hw *400 512 u8.0 weU-hprac 硬件支持的 urboot_m328p_1s_autobaud_uart0_r...
_u4_pr
414 512 u8.0 weU-jPrac 向量/spmready
urboot_m328p_1s_autobaud_uart0_r...
avrdude> 写入Flash urboot:autobaud_list_ee_ce_u4
选择尺寸用途与功能
类型
规范文件名
_hw *400 512 u8.0 weU-hprac 硬件支持的 urboot_m328p_1s_autobaud_uart0_rxd0...
_pr
414 512 u8.0 weU-jPrac 向量/spmready
urboot_m328p_1s_autobaud_uart0_rxd0...

在所有列表中，带星标的Bootloader是那些功能最丰富的Bootloader
使用的Flash空间，出于必要，必须是Flash页面的倍数或最小
引导部分。这些可以选择具有最佳功能的。显示功能显示
无需实际写入即可写入的Bootloader的属性；去除
然后 show 会将Bootloader写入Flash，并使用 -U 设置必要的熔丝：

⊠
$ avrdude -qq -c dryrun -p m328p -U urboot:autobaud_show_best_ee
378 384 u8.0 weU-jPrac 矢量/spmready urboot_m328p_1s_autobaud_uart0_rxd0_txd1_no...
$ avrdude -qq -c dryrun -p m328p -U urboot:autobaud_best_ee && echo OK
好的

第 5 章：自动生成的文件
64
与 urboot 项目一样，AVRDUDE 目前只提供经典部分的Bootloader。
下表列出了可能的功能。
2秒
WDT 超时：250ms、500ms、1s（默认）、2s、4s 或 8s（*0）
自动波特率
Bootloader在 MCU 功能范围内适应主机波特率 (*1)
串口<n>
硬件UART编号，例如，uart0（默认），uart1，...
替代<n>
替代 UART I/O 线 (*2)
9.6k波特
或者其他合理的波特率；也接受波特率单位
16兆赫
或者其他CPU频率；还接受 kHz 和 Hz 单位
[xa-q]
可选 F CPU 前缀指示符，例如 i9.2MHz (*3)
- x：外部振荡器（默认）
-i：内部振荡器
- [a-h]：内部振荡器慢 10% (a) 至 1.25% (h)
- [j-q]：内部振荡器，速度快于 1.25% (j) 至 10% (q)
斯威奥
软件 I/O，必须指定 rx 和 tx 引脚
rx[a-h][0-7]
MCU swio 接收引脚，例如 rxb0
tx[a-h][0-7]
swio 的 MCU 传输引脚，例如 txb1
莱德诺普
如果未指定 LED，则生成模板Bootloader
无 LED/Noled
除非指定了 LED，否则丢弃闪烁代码
led[+-][a-h][0-7]
生成极性为 +/- 的活动 LED 代码，例如 led+b5
双
双启动；必须为外部 SPI Flash指定 CS 引脚 (*4)
CS[a-h][0-7]
用于双引导的片选引脚，例如 csd5
硬件
生成带有硬件引导部分的Bootloader (*5)
v<n>
矢量Bootloader的可选矢量，例如 v25 或 vspmready
伊
生成支持 EEPROM r/w 的Bootloader
CE
生成可以模拟整片擦除的Bootloader
公关

生成具有复位向量保护的向量Bootloader (*6)
u1
生成跳过冗余Flash页面写入的Bootloader (*7,8)
u2
...并在模拟 CE 期间跳过冗余Flash页面擦除 (*7,8)
u3
...并在页写入期间跳过不需要的Flash页擦除 (*7,8)
u4
...并在页擦除后跳过空Flash页写入 (*8)
序列号=abc123
将序列号（例如 abc123）放在未使用的BootloaderFlash顶部
填充=urboot\x20
使用参数重复填充其他未使用的BootloaderFlash
保存=myfile.hex
将Bootloader保存到具有所选名称的文件中 (*9)
保存
将Bootloader保存到具有规范文件名的文件中 (*9)
标签=myfile.tag
使用所选名称将符号保存到标记文件 (*9)
标签
将符号保存到具有规范文件名的标记文件 (*9)
配置
显示所需的熔丝配置，但不写入存储器
显示
显示Bootloader功能但不写入Flash
清单
列出可能的Bootloader配置，但不写入Flash
最好的
选择最小的功能丰富的Bootloader（列表中的第一个），如果波特率
UART 速率错误太高，切换到 swio
帮助
显示此帮助消息并返回
注释。也可以像规范文件名的元素一样指定功能。详情
有关 urboot Bootloader及其功能，请参阅 https://github.com/stefanrueger/urboot。
*0
某些部件不提供 4 秒或 8 秒看门狗超时
*1
自动波特仅适用于 UART I/O，其中 RX 端口引脚可位寻址

第 5 章：自动生成的文件
65
*2
从经典器件来看，只有 ATtiny441/841 具有备用 UART 信号
*3
外部振荡器 (x) 之间存在细微差别，它们是合理的
准确的内部振荡器（a...h、i、j...q），对于
经典部分。前者可以承受高达 2.2% 的更高波特率误差，而对于
后者 AVRDUDE 在 0.7% 波特率错误的下限阈值处发出警告。卡西克
UART 具有整数波特率除数，这可以导致高波特率量化
错误。与最佳功能AVRDUDE一起使用可以自动替换UART I/O
当前者表现出过高的波特率错误时，将其与 swio I/O 代码一起使用。对于
也就是说，AVRDUDE 更好地了解驱动电路板的振荡器类型（x 或 i）。对于
在内部振荡器上运行的单个板上的最低波特率误差
最好测量F CPU并使用带有i前缀的测量值。或者，
可以使用标称内部 F CPU（例如 8 MHz）并使用前缀字母
使Bootloader工作：取决于字母 AVRDUDE 减去或
添加到标称 F CPU 倍数 1.25%。
*4
缺乏标准 SPI 通信的部件不支持双启动
*5
并非所有部件都提供硬件Bootloader支持； hw 使 pr 毫无意义
*6
仅当Flash大小为 2 的幂（非 ATmega406）时，复位向量保护才可用
*7
u1...u3 只是建议性的，即可以导致 u1...u4 中的任何一个
*8
u1...u4 不适用于具有 4 页擦除特性的部件
*9
该文件仍会与配置、显示或列表一起写入磁盘。引导装载程序
文件默认以 Intel Hex 格式编写并带有注释。文件名 -
将文件写入标准输出。如果Bootloader是矢量Bootloader，则Bootloader
文件包含一个用于初始化复位向量以指向Bootloader的段。标签
文件可以与终端 disasm 命令结合使用。
波特率量化误差用-v 显示。考虑 0.79% 的波特率误差
对于外部振荡器来说还可以，但是对于内部振荡器来说太高了。因此，AVRDUDE 选择
当 uart 和 swio 均未明确请求时软件 I/O：

```
$ avrdude -qv -c dryrun -p m328p -U urboot:x8mhz_56kbaud |& grep -i 波特率
波特率误差 -0.79%（外部振荡器正常）
$ avrdude -qv -c dryrun -p m328p -U urboot:i8mhz_56kbaud |& grep -i 波特率
切换到 SWIO，因为波特率误差 -0.79% 对于内部振荡器来说太高
波特率误差 -0.10%（内部振荡器正常）
```

当用户请求硬件 UART I/O 时会出现警告：

```
$ avrdude
```

-qvc dryrun -p m328p -U urboot:i8mhz_56kbaud_uart0 |& grep -i 波特率
警告：int 振荡器的高波特率误差 -0.79%：考虑切换到 swio

当 UART 量化错误时，请求最佳使 AVRDUDE 切换到 swio
认为太高：

⊠
$ avrdude -qvc dryrun -p m328p -U urboot:i8mhz_56kbaud_uart0_best |& grep -i baud.rat
切换到 SWIO，因为波特率误差 -0.79% 对于内部振荡器来说太高
波特率误差 -0.10%（内部振荡器正常）

第 5 章：自动生成的文件
66
或者，用户可以在 UART 使用的那些 rx/tx 线上请求 swio：

⊠
$ avrdude -qvc dryrun -p m328p -U urboot:i8mhz_56kbaud_uart0_swio |& grep -i baud.rat
波特率误差 -0.10%（内部振荡器正常）

标签文件可以与 disasm 结合使用：

⊠
$ avrdude -qqc dryrun -p m328p -U urboot:m328p_autobaud_ee_tags=/tmp/x \
-T“disasm -gt=/tmp/x flash 0 -1”
[...]
.equ
io.mcusr，
0x34
.equ
io.spmcsr,
0x37
.equ
内存.wdtcsr，0x60
[...]
.text
主要：
__向量重置：
L0000:
rjmp
.-386
; L7e80（urboot）
标记：
; Urboot 标记字节
L0002：
.byte
0x75、0x72
;你的
L0004：
.填充
16190, 2, 0xffff
;来自 L0000 的 Rjmp
乌尔引导：
;Bootloader的入口点
L7e80：
清除率
r1
L7e82：
在
r2，io.mcusr
L7e84：

出
io.mcusr，r1
L7e86：
LDI
r24, 0x00
; 0
L7e88：
回拨
设置看门狗
; L7f5c
L7e8a：
sbrs
r2, 1
;位 1 = 0x02
应用：
L7e8c：
rjmp
.+470
; L0064（__vector_spm_ready）
串行启动：
;串口引导加载
L7e8e：
LDI
r24, 0x0e
; 14
L7e90：
回拨
设置看门狗
; L7f5c
[...]
标签27：
L7fea：
清除率
r1
L7fec：
雷特
未使用：
; _serialno= 和/或 _fill= 为 12 个字节
L7费用：
.填充
6, 2, 0xffff
npages_vecnum：
;使用3页；矢量/spmready Bootloader
L7ffa:
.byte
0x03、0x19
; __
L7ffc:

rjmp
pgm_write_page
; L7f78
功能版本：
;编码 u8.0 weU-jPra-
L7ffe：
.byte
0xe6、0x40
; _

第 5 章：自动生成的文件
67
值得注意的是 -c dryboot 允许上传一个 urboot Bootloader，然后
从点开始，模拟 -c urclock -x nometadata。这意味着，例如 -c dryboot 不会
允许覆盖Bootloader：

⊠
$ avrdude -q -c dryboot -p m328p -U urboot:autobaud_ee -U flash.hex \
-T“写入Flash0x7f80 0x00”
处理 -U flash:w:urboot:autobaud_ee:i
从输入文件 urboot:autobaud_ee 读取Flash的 376 字节
将 376 字节写入Flash，写入 376 字节，验证 376 字节
设置Bootloader urboot:autobaud_ee 的熔丝位
在 [0x7e80, 0x7fff] 中检测到 urboot Bootloader u8.0 weU-jPra-，向量=25
处理 -U flash:w:blink.hex:i
从输入文件blink.hex读取Flash的354字节
将 354 字节写入Flash，写入 354 字节，验证 354 字节
处理-T写Flash0x7f80 0x00
警告：（写）编程器写保护Flash地址 0x7f80
阿夫杜德完成了。
谢谢。

更重要的是，如果满足以下条件，-c dryboot 会修补任何要上传的草图的向量表：
之前安装的Bootloader是一个 urboot 矢量Bootloader，就像 avrdude -c urclock
-x nometadata 会；这是矢量Bootloader工作所必需的（有关详细信息，请参阅
https://github.com/stefanrueger/urboot 项目）：

⊠
$ avrdude -qq -c dryboot -p m328p -U urboot:autobaud_ee -U flash.hex \
-T“disasm flash 0 2” -T“disasm flash 100 4”
L0000：3f 比照
rjmp
.-386
; L7e80
L0064：0c 94 34 00
跳跃
0x0068

相反，使用 -c dryrun 不会修补草图的向量表，因为它会模拟和
行为就像外部编程器：

⊠
$ avrdude -qq -c dryrun -p m328p -U urboot:autobaud_ee -U flash.hex \
-T“disasm flash 0 2” -T“disasm flash 100 2”
L0000: 33 c0
rjmp
.+102
; L0068
L0064：11 c0
rjmp

.+34
; L0088

以下示例准备了一个修补后的草图，以便通过自己编写的上传器上传
或者通过 urboot 的双启动。请注意，385 是Bootloader的大小加 1，并且
生成的文件blink-patched.hex的大小为flash负384。
选项 -A AVRDUDE 从修补的草图中删除所有尾随 0xff。

⊠
$ avrdude -qq -c dryboot -p m328p -U urboot:autobaud_ee -U flash.hex \
-T“保存Flash0 -385眨眼补丁.十六进制：I”

68
6 编程器特定信息
6.1 爱特梅尔STK600
相应的 STK600 路由和插槽卡支持以下设备：
路由卡
插口卡
设备
STK600-ATTINY10
t4 t5 t9 t10
STK600-RC008T-2
STK600-DIP
t11 t12 t13 t13a t25 t45 t85
STK600-RC008T-7
STK600-DIP
t15
STK600-RC014T-42
STK600-SOIC
t20
STK600-RC020T-1
STK600-DIP
t2313 t2313a t4313
STK600-TinyX3U
t43u
STK600-RC014T-12
STK600-DIP
t24 t44 t84 t24a t44a
STK600-RC020T-8
STK600-DIP
t26 t261 t261a t461 t861 t861a
STK600-RC020T-43
STK600-SOIC
t261 t261a t461 t461a t861 t861a
STK600-RC020T-23
STK600-SOIC
t87 t167
STK600-RC028T-3
STK600-DIP
t28
STK600-RC028M-6
STK600-DIP
t48 t88 m8 m8a m48 m88 m168 m48p
m48pa m88p m88pa m168p m168pa
m328p
QT600-ATTINY88-QT8
t88
STK600-RC040M-4
STK600-DIP

m8515 m162
STK600-RC044M-30
STK600-TQFP44
m8515 m162
STK600-RC040M-5
STK600-DIP
m8535 m16 m16a m32 m32a m164p
m164pa m324p m324pa m644 m644p
m644pa m1284p
STK600-RC044M-31
STK600-TQFP44
m8535 m16 m16a m32 m32a m164p
m164pa m324p m324pa m644 m644p
m644pa m1284p
QT600-ATMEGA324-QM64
m324pa
STK600-RC032M-29
STK600-TQFP32
m8 m8a m48 m88 m168 m48p m48pa
m88p m88pa m168p m168pa m328p
STK600-RC064M-9
STK600-TQFP64
m64 m64a m128 m128a m1281 m2561
c32 c64 c128
STK600-RC064M-10
STK600-TQFP64
m165 m165p m169 m169p m169pa
m325 m325p m329 m329p m645 m649
m649p
STK600-RC100M-11
STK600-TQFP100
m640 m1280 m2560
STK600-ATMEGA2560
米2560
STK600-RC100M-18
STK600-TQFP100
m3250 m3250p m3290 m3290p m6450
米6490
STK600-RC032U-20
STK600-TQFP32
USB82 USB162 M8U2 M16U2 M32U2
STK600-RC044U-25
STK600-TQFP44
m16u4 m32u4
STK600-RC064U-17
STK600-TQFP64
m32u6
USB646

USB1286
USB647
USB1287
STK600-RCPWM-22
STK600-TQFP32
m32c1 m64c1 m16m1 m32m1 m64m1

第 6 章：编程器特定信息
69
STK600-RCPWM-19
STK600-SOIC
pwm2 pwm3 pwm2b pwm3b pwm216
脉宽调制316
STK600-RCPWM-26
STK600-SOIC
脉宽调制81
STK600-RC044M-24
STK600-TSSOP44
m16hvb m32hvb
STK600-HVE2
m64hve
STK600-ATMEGA128RFA1
m128rfa1
STK600-RC100X-13
STK600-TQFP100
x64a1 x128a1 x128A1revd x128a1u
STK600-ATXMEGA1281A1
x128a1
QT600-ATXMEGA128A1-
QT16
x128a1
STK600-RC064X-14
STK600-TQFP64
x64a3 x128a3 x256a3 x64d3 x128d3
x192d3 x256d3
STK600-RC064X-14
STK600-MLF64
x256a3b
STK600-RC044X-15
STK600-TQFP44
x32a4 x16a4 x16d4 x32d4
STK600-ATXMEGAT0
x32t0
STK600-uC3-144
uc3a0512 uc3a0256b uc3a0128b
STK600-RCUC3A144-33
STK600-TQFP144
uc3a0512 uc3a0256b uc3a0128b
STK600-RCuC3A100-28
STK600-TQFP100
uc3a1512b uc3a1256b uc3a1128b
STK600-RCuC3B0-21
STK600-TQFP64-2
uc3b0256b uc3b0512revcb uc3b0512b

uc3b0128b uc3b064b uc3d1128b
STK600-RCuC3B48-27
STK600-TQFP48
uc3b1256b uc3b164b
STK600-RCUC3A144-32
STK600-TQFP144
uc3a3512b
uc3a3256b
uc3a3128b
uc3a364b
uc3a3256sb
uc3a3128sb
uc3a364sb
STK600-RCUC3C0-36
STK600-TQFP144
uc3c0512b
uc3c0256b
uc3c0128b
uc3c064b
STK600-RCUC3C1-38
STK600-TQFP100
uc3c1512b
uc3c1256b
uc3c1128b
uc3c164b
STK600-RCUC3C2-40
STK600-TQFP64-2
uc3c2512b
uc3c2256b
uc3c2128b
uc3c264b
STK600-RCUC3C0-37
STK600-TQFP144
uc3c0512b
uc3c0256b
uc3c0128b
uc3c064b
STK600-RCUC3C1-39
STK600-TQFP100
uc3c1512b
uc3c1256b
uc3c1128b
uc3c164b
STK600-RCUC3C2-41
STK600-TQFP64-2
uc3c2512b
uc3c2256b
uc3c2128b

uc3c264b
STK600-RCUC3L0-34
STK600-TQFP48
uc3l064b uc3l032b uc3l016b
QT600-AT32UC3L-QM64
uc3l064b
在给 STK600 上电之前，请确保安装了正确的插座和路由卡。
STK600固件确保安装的插座和路由卡相互匹配
（使用内部存储在非易失性存储器中的表），它无法处理以下情况：
使用了错误的路由卡，例如。 g。路由卡 STK600-RC040M-5（这意味着
对于具有 ADC 的 40 引脚 DIP AVR，电源引脚位于
使用了 ATmega8515（使用“行业标准”引脚排列）
Vcc 和 GND 位于对角）。

第 6 章：编程器特定信息
70
请注意，对于使用路由卡 STK600-RC008T-2 的设备，为了使用 ISP
模式下，必须移除 AREF0 的跳线，否则会阻塞 ISP 之一
信号。即使安装了该跳线，也可以使用高压串行编程。
STK600 的 ISP 系统包含针对快捷方式和其他接线的检测
错误。 AVRDUDE 在尝试进入 ISP 编程之前启动连接检查
模式，如果目标未准备好进行 ISP 编程，则显示结果。
高压编程要求目标电压至少设置为 4.5 V，以便
去工作。这可以使用终端模式来完成，请参阅第 3 章 [终端模式操作]，
第 36 页。
6.2 使用 FLIP 版本 1 的 DFU Bootloader
使用 FLIP 协议版本 1 的Bootloader会遇到一些非常具体的行为。
这些Bootloader没有选择访问Flash和 EEP 以外的存储区域
ROM。
当Bootloader启动时，它会进入安全模式，其中唯一可接受的
access是查询设备配置参数（用于签名）
AVR 设备）。离开此模式的唯一方法是整片擦除。由于整片擦除通常是
重新编程Flash时，由 -U 选项暗示，这种特性可能不是很明显
立即显而易见。
有时，已禁用安全模式的Bootloader似乎不再响应
具有合理的配置数据，但所有查询都只有 0xFF。当使用这些查询时
要获得相当于签名的内容，AVRDUDE 在这种情况下只能继续
强制使用 -F 选项覆盖签名检查。
整片擦除可能不会擦除 EEPROM，至少在某些版本的
Bootloader。
6.3 串行UPDI编程器
SerialUPDI 编程器可用于使用非常简单的方式对仅 UPDI 设备进行编程
ple串行连接。您可以在这里阅读更多有关详细信息 https://github.com/
SpenceKonde/AVR-Guidance/blob/master/UPDI/jtag2updi.md
SerialUPDI 编程器已使用 FT232RL USB->UART 接口进行了测试
以下连接布局（从上面链接的 Spence Kohde 页面复制）：

至目标设备
数据传输|
_____
Rx |----------,----------| UPDI---\/\/------->
Tx---/\/\/\---Tx |----|<|---'
.--------|地线
470欧姆
电阻器
Vcc|----------|电源电压
1k
华视|
.`
|_____
接地|----------'

当前 SerialUPDI/AVRDUDE 集成有几个限制，列出如下：
低。
目前可用的器件仅支持 UPDI NVM 编程模型 0、2、3 和 5，
但模型 4 也有实验性的实现 - 它仅在

第 6 章：编程器特定信息
71
单个设备，因此预计其他设备会出现问题。完整的 NVM v4 模式支持
一旦硬件广泛可用就提供。
AVRDUDE 的核心功能之一是通过读取设备验证连接
在任何操作之前签名，但此操作在 UPDI 锁定设备上是不可能的。
因此，为了能够连接到这样的设备，您必须提供 -F 来覆盖此设置
检查。
请注意：在对锁定设备进行写操作期间使用 -F 将强制整片擦除。使用
仔细地。
您可能会注意到的另一个问题是使用串行写入 EEPROM 的性能缓慢
用于 AVR Dx 设备的 alUPDI。这可以通过更改 avrdude.conf 部分来解决
该设备 - 将 EEPROM 页面大小更改为 0x20（而不是默认的 1），如下所示：
#----------
AVR128DB28
#----------
部分父母
“.avrdx”
编号
=“avr128db28”；
描述
=“AVR128DB28”；
签名= 0x1E 0x97 0x0E；
存储器“闪现”
尺寸
= 0x20000；
偏移量
= 0x800000；
页大小=0x200；
读取大小
= 0x100；
;
内存“eeprom”
尺寸
= 0x200；
偏移量
= 0x1400；
页大小=0x20；
读取大小
= 0x100；
;
;
USERROW 的目的是提供将配置详细信息写入已经存在的能力
锁定设备，目前SerialUPDI接口支持此功能。请注意：在
锁定设备，写入时无法读回 USERROW 内容，因此
自动验证很可能会失败，为了防止出现错误消息，请使用 -V。
如果您遇到 SerialUPDI 接口问题，请确保运行

启用调试输出的预期命令（-v -v -v）并提供此详细输出-
与您的错误报告放在一起。您还可以尝试使用 pymcuprog 执行相同的操作
(https://github.com/microchip-pic-avr-tools/pymcuprog) 实用程序，带有 -v debug 和
也提供它的输出。您会注意到两个输出非常相似，这是
故意这样实现 - 它应该对 UPDI 协议进行分析
怪癖更容易。

第 6 章：编程器特定信息
72
6.4 编程器 LED 管理
一些硬件编程器有 LED，固件可以完全控制它们，无需
AVRDUDE 有办法影响其 LED 状态。其他编程器有 LED
并期望主机下载器/上传器能够处理它们，例如 bit-banging pro-
语法师、基于 ftdi 的编程器或 linuxgpio 编程器。
对于那些编程器来说
AVRDUDE 支持四个 LED（RDY、ERR、PGM 和 VFY）
通过相应 -c 编程器代码中相应的子例程进行设置。
一旦编程器初始化，RDY LED 就会被设置，并在 AVR- 时关闭。
杜德退出。在读取、写入或擦除目标期间，PGM LED 闪烁
大约 2.5 Hz，同时 VFY LED 在书面确认验证期间亮起
帐篷。错误通过 ERR LED 指示。
假设 AVRDUDE 达到了可以访问 LED 的程度，并且 RDY LED
然后打开，退出时，AVRDUDE 将使 LED 处于以下状态：
铂族金属VFY
错误语义
关闭
关闭
关闭
OK：所有任务均已完成，没有错误
关闭
关闭
上
一些与读、写或擦除无关的错误
上
关闭
上
读、写或擦除错误
关闭
上
上
验证错误，但没有读、写或擦除错误
上
上
上
验证错误和读、写或擦除错误
退出后不应显示其他组合。

73

附录 A 平台相关信息

A.1 Unix

A.1.1 Unix 安装

最新安装提示请参考https://github.com/avrdudes/avrdude/wiki。

A.1.2 Unix 配置文件

当使用默认的 --prefix 配置选项构建 AVRDUDE 时，默认的配置-
Unix 系统的配置文件位于 /usr/local/etc/avrdude.conf。这可以是
使用 -C 命令行选项覆盖。此外，用户的主目录是
搜索名为 .avrduderc 的文件，如果找到，则用于增强系统默认值
配置文件。

A.1.2.1 FreeBSD 配置文件

当使用FreeBSD ports系统安装AVRDUDE时，系统配置
文件始终是/usr/local/etc/avrdude.conf。

A.1.2.2 Linux 配置文件

当使用 rpm 软件包安装 AVRDUDE 时，系统配置文件将
始终为 /etc/avrdude.conf。

A.1.3 Unix 端口名称

并行和串行端口设备文件名是特定于系统的。 macOS 没有默认值
串行端口名称，但可以在 /dev/cu.* 下找到可用端口。请注意
AVRDUDE 不支持 macOS 下的并口编程。
下表列出了给定系统的默认名称。

系统	默认并行端口	默认串口
自由BSD	/dev/ppi0	/dev/cuad0
Linux	/dev/parport0	/dev/ttyS0
索拉里斯	/dev/打印机/0	/dev/term/a

在 FreeBSD 系统上，AVRDUDE 使用 ppi(4) 接口来访问并行
端口和用于串行端口访问的 sio(4) 驱动程序。
在 Linux 系统上，AVRDUDE 使用 ppdev 接口来访问并行端口
以及用于串行端口访问的 tty 驱动程序。
在 Solaris 系统上，AVRDUDE 使用 ecpp(7D) 驱动程序访问并行端口
以及用于串行端口访问的 asy(7D) 驱动程序。

A.1.4 Unix USB 权限

在大多数情况下，内核驱动程序会初始化一个即插即用设备，使其归用户 root 所有。
和 root 用户组仅具有读/写权限，导致设备无法访问
给普通用户。虽然用户可以以 root 身份运行 AVRDUDE 会话，但这绝对不好
练习。为 USB 即插即用设备提供正确的权限会好得多。 USB

附录 A：平台相关信息
74
AVR 编程器通常由两个字节的十六进制供应商 ID 和一个两个字节来标识
字节十六进制产品 ID。两者通常都用于识别需要新的设备
权限。
A.1.4.1 FreeBSD USB 权限
在 FreeBSD 中， /usr/local/etc/devd 中所谓的 devd 配置文件用于修改权限 -
插入的 USB 设备的版本。以下是 Atmel 的 JTAGICE3 编程器的示例
（产品 ID 0x2110 或 0x2140）由 Atmel（供应商 ID 0x0eb）可以给出适当的 per-
使用文件 jtagice3.conf 的任务：

```
通知 100 {
匹配“系统”“USB”；
匹配“子系统”“设备”；
匹配“类型”“附加”；
匹配“供应商”“0x03eb”；
匹配“产品”“(0x2110|0x2140)”；
操作“chmod 660 /dev/$cdev”；
操作“chgrp yourgroup /dev/$cdev”；
};
```

yourgroup 将是一个组，用户应该是希望访问该组的成员
给编程器。
A.1.4.2 Linux USB 权限
Linux 有一个特殊的用户空间 /dev 设备管理器，称为 udev，它处理以下内容：
其他，即插即用 USB 设备。建议指定所谓的 udev 规则
而是定义这些设备的访问权限。这些规则通常驻留在一个文件中
名称为 nn-descriptive-name.rules，位于目录 /etc/udev/rules.d 中。在这里，n
是一个两位数，用于确定 udev 规则文件的词汇顺序
已处理。后面处理的规则可以覆盖前面的规则，但不建议放置
用户生成的规则高于 60，因为它们需要的一些操作由
更高级别的系统规则。
这是一个典型的 udev 规则，允许普通用户访问插入的 AVRISP
Atmel 的 mkII 编程器（产品 ID 0x2104）（供应商 ID 0x0eb）：

```
子系统==“usb”，ATTRS {idVendor}==“03eb”，ATTRS {idProduct}==“2104”，\
模式=“0660”，标签+=“uaccess”
```

这为相应的设备节点提供了 0660 访问权限：这意味着读/写
对于 root 用户和属于设备组的任何用户，设备驱动程序
可能会分配给与默认根不同的组。规则的关键是附件
名为 uaccess 的 TAG，其作用是登录守护程序应用动态用户
设备节点的访问控制列表，使设备可供当前登录的用户使用
用户。当在愤怒中使用时，udev规则必须出现在一行；上面的例子被破坏了
分成两行，以便它适合示例框中。
AVRDUDE 的开发者选项 -cprogrammer/u 将显示上面建议的 udev 规则
对于指定的编程器。允许使用通配符：

附录 A：平台相关信息
75

⊠
$ avrdude -c jtag*/u
1. 检查下面建议的 udev 规则；安装运行：
avrdude -c“jtag*/u”|尾-n +11 | sudo tee /etc/udev/rules.d/55-avrdude-jtagX.rules
须藤 chmod 0644 /etc/udev/rules.d/55-avrdude-jtagX.rules
2. 拔下所有 AVRDUDE USB 编程器并重新插入
3. 享受用户对 USB 编程器的访问
注意：要安装 AVRDUDE 已知的所有 udev 规则，请执行以下操作：avrdude -c "*/u" |更多
由 avrdude -c "jtag*/u" 生成
ACTION！=“添加|更改”，GOTO=“avrdude_end”
jtag2dw、jtag2fast、jtag2、jtag2isp、jtag2pdi、jtag2slow、jtagmkII、jtag2avr32
子系统==“usb”，ATTRS {idVendor} ==“03eb”，ATTRS {idProduct}==“2103”，\
模式=“0660”，标签+=“uaccess”
jtag3、jtag3dw、jtag3isp、jtag3pdi、jtag3updi
子系统==“usb”，ATTRS {idVendor}==“03eb”，ATTRS {idProduct}==“2110”，\
模式=“0660”，标签+=“uaccess”
KERNEL==“hidraw*”，SUBSYSTEM==“hidraw”，ATTRS {idVendor}==“03eb”，\
ATTRS{idProduct}==“2110”，MODE =“0660”，TAG + =“uaccess”
子系统==“usb”，ATTRS {idVendor} ==“03eb”，ATTRS {idProduct}==“2140”，\
模式=“0660”，标签+=“uaccess”
KERNEL==“hidraw*”，SUBSYSTEM==“hidraw”，ATTRS {idVendor}==“03eb”，\
ATTRS{idProduct}==“2140”，MODE =“0660”，TAG + =“uaccess”
jtagkey
子系统==“usb”，ATTRS {idVendor} ==“0403”，ATTRS {idProduct}==“cff8”，\
模式=“0660”，标签+=“uaccess”

同样，每条规则必须写成一行：将规则分成两行只是
完成以使 AVRDUDE 的输出适合盒装显示。 HID 模式下的 USB 设备需要
第二条规则涉及 hidraw 子系统，如上所示。
A.1.5 Unix 文档
AVRDUDE 安装手册页以及信息、HTML 和 PDF 文档。的
手册页安装在 /usr/local/man/man1 区域，而 HTML 和 PDF 文档-
umentation 安装在 /usr/local/share/doc/avrdude 目录中。信息手册是
安装在/usr/local/info/avrdude.info。
请注意，这些位置可以通过各种配置选项（例如 --prefix）更改。
A.2 窗口
A.2.1 安装
最新安装提示请参考https://github.com/avrdudes/avrdude/wiki。
A.2.2 Windows 配置文件

附录 A：平台相关信息
76
A.2.2.1 Windows 配置文件名
Windows 上的 AVRDUDE 会查找名为 avrdude.conf 的系统配置文件，并且
在同一目录中查找 avrdude.rc 的用户覆盖配置文件
avrdude.exe 位于。
A.2.2.2 Windows 配置文件位置
Windows 上的 AVRDUDE 有不同的方式搜索系统和用户配置
口粮文件。下面是查找配置文件的搜索方法：
1.仅
为了
的
系统
配置
文件：
<来自哪个应用程序的目录
已加载>/../etc/avrdude.conf
2. 应用程序加载的目录。
3. 当前目录。
4.Windows系统目录。
在 Windows NT 上，该目录的名称是
系统32。
5. Windows 目录。
6. PATH 环境变量中列出的目录。
A.2.3 Windows 端口名称
A.2.3.1 Windows 串行端口
当您选择串行端口时（即使用 STK500 时），请使用 Windows 串行端口
设备名称，例如：com1、com2 等。
A.2.3.2 Windows 并行端口
AVRDUDE 不支持 Windows 中的并行端口编程。如果你需要运行
AVRDUDE 在并行端口上使用编程器，您可能想尝试其中一种 BSD
或Linux。

77
附录 B 故障排除
请通过 https://github.com/avrdudes/avrdude/issues 报告遇到的任何错误。
AVRDUDE 的 wiki https://github.com/avrdudes/avrdude/wiki 是一个很好的地方
了解如何在各种平台上安装 AVRDUDE，一般来说，了解一些
交易的技巧。
特别是常见问题解答 (https://github.com/avrdudes/avrdude/
wiki/FAQ）和已知限制（https://github.com/avrdudes/avrdude/wiki/
avrdude 的已知限制 (Known-limitations-of-avrdude) 值得一读。
以下是一些可能出错的情况以及该怎么做的示例：
・ 问题：我在Windows 下使用串行编程器并收到以下错误：
avrdude:serial_open():无法设置设备“com1”的属性，
解决方案：此问题似乎出现在某些版本的 Cygwin 中。指定
“/dev/com1”而不是“com1”应该有帮助。
・ 问题：我使用的是Linux，而我的AVR910 编程器速度非常慢。
解决方案：这里有两个问题。首先，系统可能会等待一段时间
将数据从串行端口传递到程序。 Linux下执行以下命令
解决这个问题（您可能需要 root 权限）。
设置串行端口 low_latency
其次，串行接口芯片可能会延迟中断一段时间。这就是——
可以通过将 FIFO 阈值设置为 1 来更改行为。在Linux下这可以
只能通过更改 drivers/char/serial.c 中的内核源代码来完成。搜索文件
UART_FCR_TRIGGER_8 并将其替换为 UART_FCR_TRIGGER_1。请注意，总体
如果串行线路上的吞吐量较高，性能可能会受到影响。另请注意，您
修改内核的风险由您自行承担。
・ 问题：我不使用Linux，而且我的AVR910 编程器速度非常慢。
解决办法：出现这种情况的原因同上。如果您知道如何解决
这在您的操作系统上，请告诉我们。
・ 问题：使用-U 选项对EEPROM 进行页模式编程不会擦除
写入之前的 EEPROM 单元，因此不一定会覆盖非 0xff 值。
解决方案：这是 JTAG EEPROM 编程工作方式的固有特征，
并在数据表中进行了记录。为了成功编程
EEPROM，需要在 EESAVE 熔丝未编程的情况下进行事先整片擦除。这个
也适用于 STK500 和 STK600 的高压编程模式。
然而，终端认识到编程器很难写入 EEPROM。
然后读取Flash、EEPROM 以及引导行内容（如果存在），执行整片擦除
然后把这些存储器写回来。当刷新缓存或离开时会发生这种情况
终端并需要一些时间。为此，需要对 EESAVE 进行取消编程。
・ 问题：如何关闭DWEN 熔丝位?
解决办法：如果DWEN（debugWIRE使能）熔丝被激活，/RESET引脚不工作
无法正常工作，无法建立正常的 ISP 通信。有
再次停用该熔丝位的两个选项：高压编程，或获取
JTAG ICE mkII 通话 debugWIRE，并准备目标 AVR 接受正常 ISP
再次沟通。

附录 B：故障排除
78
第一个选项需要能够进行高压编程的编程器
（串行或并行，取决于 AVR 设备），例如 STK500。在
高压编程模式，/RESET 引脚最初使用 12 V 脉冲激活
（因此称为高电压），因此目标 AVR 随后可以重新编程，
并且DWEN熔丝可以被清除。通常，此操作不能在以下情况下执行：
不过，AVR 位于目标电路中。
第二个选项需要一个可以使用 debugWIRE 协议的 JTAG ICE mkII。
ICE 需要使用 JTAG-to-ISP 适配器连接到目标，因此
JTAG ICE mkII 可用作 debugWIRE 启动器以及 ISP 编程器。
然后将使用 jtag2isp 编程器类型激活 AVRDUDE。最初的
ISP 通信尝试将失败，但 AVRDUDE 随后尝试启动调试 -
有线重置。成功后，这将使目标 AVR 处于可以进行以下操作的状态：
接受标准 ISP 通信。然后 ICE 被签署（这将使它
也从 USB 注销），因此之后必须再次调用 AVRDUDE。
此时，标准ISP通信可以工作，因此DWEN熔丝可以被清除。
JTAG 至 ISP 适配器的引脚映射为：
JTAG
互联网服务供应商
1
3
2
6
3
1
4
2
6
5
9
4
• 问题：区分多个USBtinyISP（或USBasp）编程器
解决方案： -c usbtiny 编程器区分多个物理 USBtinyISP de-
基于它们的busdir:devicefile对来描述它们在USB层次结构中的位置
在特定主机上。该对可以在 -P usb:busdir:devicefile 操作中指定
。总线和设备的命名约定取决于操作系统。
Linux、FreeBSD 和 Windows 的示例分别为：
$ avrdude -c usbtiny -p atmega8 -P usb:003:025
$ avrdude -c usbtiny -p atmega8 -P usb:/dev/usb:/dev/ugen1.3
$ avrdude -c usbtiny -p atmega8 \
-P 'usb:bus-0:\\?\usb#vid_16c0&pid_05dc#0001#{a5...ed}--WinUSB'
Windows 设备名称包含反斜杠文件分隔符，因此 -P 选项将
需要在命令行上进行适当的引用，例如，在 bash 中使用单引号。
对于 USBasp，与 USBtiny 相同的 -P usb:busdir:devicefile 选择正确的
设备。
或者，USBasp 可以使用 -P 通过序列号选择设备

USB：序列号。
请注意，avrdude -v -P usb:xyz 将在总线上打印出合适的编程器：
求和 xyz 与任何设备都不匹配。
$ avrdude -v -Pusb:xyz -c usbasp -p m328p 2>&1 | grep ^找到
找到 USBasp，busdir:devicefile = 001:008，serial_number = 0001
找到 USBasp，busdir:devicefile = 001:009，serial_number = 1234
$ avrdude -qq -c USBasp -p atmega8 -P usb:34

附录 B：故障排除
79
• 问题：我做不到。。。当目标处于 debugWIRE 模式时。
解决方案：debugWIRE 模式有一些限制。
debugWIRE 协议是 Atmel 专有的单线（加接地）协议，用于
允许使用 /RESET 线对较小的 AVR 设备进行在线仿真。德-
bugWIRE 模式是通过激活 DWEN 熔丝启动的，然后重新启动
目标。
虽然此模式主要用于调试/仿真，但它还提供
编程能力有限。实际上，唯一可以读取的内存区域
或在此模式下编程的是Flash和 EEPROM。也可以读出
签名。无法访问所有其他内存区域。没有整片擦除功能
debugWIRE 模式下的功能；相反，在重新编程Flash时，每个Flash页面
在更新之前被删除。这是由 JTAG ICE mkII 透明地完成的
（或 AVR Dragon）。从 debugWIRE 模式返回的唯一方法是启动一个特殊的
JTAG ICE mkII（或 AVR Dragon）的命令序列，因此 debugWIRE
模式将暂时禁用，并且可以使用正常 ISP 访问目标
编程。该序列是使用 JTAG ICE mkII 自动启动的
或 AVR Dragon 处于 ISP 模式时，当检测到无法进入 ISP 模式时。
• 问题：我想使用JTAG ICE mkII 通过PDI 对Xmega 器件进行编程。
文档告诉我使用 JTAGICE mkII 的 XMEGA PDI 适配器
应该随套件一起发货，但我没有。
解决方案：使用以下引脚映射：
JTAG ICE
目标
鱿鱼缆
PDI
MKII探针
针脚
颜色
标头
1（TCK）
黑色
2（接地）
接地
白色
6
3（TDO）
灰色
4（VT 参考）
参考电压
紫色
2
5（经颅磁刺激）
蓝色
6（nSRST）
PDI时钟

绿色
5
7（北卡罗来纳州）
黄色
8（nTRST）
橙色
9（TDI）
PDI数据
红色
1
10（接地）
棕色
• 问题：我想使用我的AVR Dragon 通过PDI 对Xmega 设备进行编程。
解决方案：使用 Dragon 上的 6 针 ISP 接头以及以下针脚映射：
龙
目标
ISP 标头
针脚
1（SDI）
PDI数据
2（VCC）
电压控制电路
3（SCK）
4（标准数据输出）
5（重置）
PDI 时钟/RST
6（接地）
接地
• 问题：我想使用 AVRISP mkII 对 ATtiny4/5/9/10 设备进行编程
通过 TPI。如何连接引脚？
解决方案：使用以下引脚映射：

附录 B：故障排除
80
AVRISP
目标
阿蒂尼
连接器
针脚
引脚#
1（SDI）
TPI数据
1
2（VT 参考）
电源电压
5
3（SCK）
TPI时钟
3
4（标准数据输出）
5（重置）
/重置
6
6（接地）
接地
2
• 问题：我想使用串行/并行 bitbang 对 ATtiny4/5/9/10 设备进行编程
编程器。如何连接引脚?
解决方案：由于TPI只有1个引脚用于双向数据传输，所以SDI和SDO
引脚应连接到 ATtiny 器件上的 TPIDATA 引脚。然而，1K
电阻应放置在 SDO 和 TPIDATA 之间。
SDI 引脚连接到
直接TPIDATA。 SCK 引脚连接到 TPICLK。
此外，Vcc、/RESET 和 GND 引脚应连接到各自的端口
在 ATtiny 设备上。
• 问题：如何使用 FTDI FT232R USB 转串口设备进行 bitbang 程序 -
明?
解决方案：当将FT232直接连接到目标Atmel设备的引脚时，
编程器定义中定义的引脚极性应通过预反转
修复波形符。例如，dasa编程器连接时会是这样的
通过 FT232R 设备（注意引脚 7、4、3 和 8 前面的波浪号）：
编程器
编号
=“dasa_ftdi”；
描述
=“串口敲击，重置=rts sck=dtr sdo=txd sdi=cts”；
类型
= 塞尔维亚语；
重置=~7；

sck
=~4；
sdo
=~3；
sdi
=~8；
;
注意这里使用的是FT232设备作为普通串口，而不是使用FTDI
驱动程序处于特殊的 Bitbang 模式。
・ 问题：我的 ATtiny4/5/9/10 读数正常，但任何对其编程的尝试（通过
TPI）失败。相反，内存保留旧内容。
解决方案：注意这些器件的有限编程电源电压范围。
通过 TPI 进行在线编程仅由数据表在 Vcc = 5 V 时保证。
・ 问题：我的ATxmega。 。 .A1/A2/A3 无法使用我的 PDI 进行编程
AVR龙.不过，通过 JTAG ICE mkII 进行编程是可行的，亲们也是如此。
通过JTAG编程。
解决方案：目前还没有（2010 年第一季度）。
据说 AVR Dragon 只能对 A4 Xmega 子设备进行编程
家庭。

附录 B：故障排除
81
・ 问题：使用 AVRISPmkII 或 STK600 编程时，AVRDUDE 挂起
当对特定大小（例如 246 字节）的文件进行编程时。其他（更大或更小）
不过尺寸可以。
解决方案：这是由于错误处理零长度数据包 (ZLP) 导致的错误
在某些版本的 libusb 0.1 API 包装器中，该包装器随 libusb 1.x 一起提供
Linux 发行版。所有内核版本 < 2.6.31 且 libusb >= 的 Linux 系统
据报道 1.0.0 < 1.0.3 受此影响。
另请参阅：http://www.libusb.org/ticket/6
・ 问题：刷新固件后会降低目标的时钟速度（例如通过
CLKPR 寄存器），进一步的 ISP 连接尝试失败。或者编程器不能
初始化与全新芯片的通信。
解决方案：即使 ISP 启动时将 /RESET 拉低，目标仍继续运行
以内部时钟速度由之前运行的固件定义或设置
由工厂。因此，必须适当降低ISP时钟速度（降低到更低）
小于内部时钟速度的 1/4）在 ISP 初始化之前使用 -B 选项
序列将会成功。
由于这会减慢整个后续 ISP 会话的速度，因此仅发出
使用慢速 ISP 时钟（选项 -e）进行整片擦除，然后在
更高的速度。可以在那里使用选项 -D，以防止另一个不需要的擦除周期。

附录 C 编程器名单

AVRDUDE 支持以下编程器：左栏列出编程器的 id 为
用于 -c，而右列包含简短描述和可用列表
括号内为编程接口；请参见第 4.2 节 [编程器定义]，第 54 页。
AVRDUDE 配置文件中有关于每个编程器的更多详细信息。

2232hio
2232hio 基于 FT2232H，带缓冲器和 LED（TPI、ISP）
4232小时
FT4232H编程器（TPI、ISP）
89ISP
Atmel at89isp 电缆（TPI、ISP）
abcmini
ABCmini Board，又名 Dick Smith HOTCHIP（TPI、ISP）
阿达水果_宝石
Trinket Gemma Bootloader伪装成 USBtiny (SPM)
阿尔夫
Nightshade ALF-PgmAVR 通过 PC 并行端口（TPI、ISP）
阿尔杜伊诺
使用 STK500 v1 协议 (SPM) 的 Arduino Bootloader
arduino-ft232r
Arduino：FT232R连接到ISP（TPI，ISP）
迪西米拉
Arduino：FT232R连接到ISP（TPI，ISP）
arduino_as_isp
AVR 作为带有 Arduino-as-ISP FW (ISP) 的编程器
arduino_gemma
Arduino Gemma Bootloader伪装成 USBtiny (SPM)
arduinoisp
Arduino品牌USBtiny ISP编程器（TPI、ISP）
阿尔杜伊诺斯波格
Arduino品牌USBtiny ISP编程器（TPI、ISP）
阿蒂斯普
用于 AVR-SDK1 的 AT-ISP v1.1 编程电缆（TPI、ISP）
阿特梅利切
Atmel-ICE（JTAG、XMEGAJTAG、AVR32JTAG）
atmelice_jtag
Atmel-ICE（JTAG、XMEGAJTAG、AVR32JTAG）
atmelice_dw
Atmel-ICE（调试WIRE）
atmelice_isp
爱特梅尔-ICE (ISP)
atmelice_pdi
爱特梅尔-ICE (PDI)
atmelice_tpi
爱特梅尔-ICE (TPI)
atmelice_updi

爱特梅尔-ICE (UPDI)
AVR109
Atmel Bootloader（AVR109、AVR911）（SPM）
AVR911
Atmel Bootloader（AVR109、AVR911）（SPM）
AVR910
Atmel 低成本串行编程器 (ISP)
阿夫夫迪
FT2232H/D 编程器（TPI、ISP）
2232小时
FT2232H/D 编程器（TPI、ISP）
阿夫里斯普
使用 STK500 的串行 Atmel AVR ISP (ISP)
avrisp-u
神田 AVRISP-U (TPI, ISP)
阿夫里斯姆克II
USB Atmel AVR ISP mkII（TPI、ISP、PDI）
AVRISP2
USB Atmel AVR ISP mkII（TPI、ISP、PDI）
avrispv2
串行 Atmel AVR ISP（TPI、ISP）
巴斯科姆
Bascom SAMPLE 编程电缆（TPI、ISP）
冲击波
Altera ByteBlaster（TPI、ISP）
BSD
Brian S. Dean 的并行编程器（TPI、ISP）
丁螺酯
AVR 编程模式 (ISP) 下的总线海盗
巴士海盗_bb
Bitbang 模式下的总线海盗（TPI、ISP）
蝴蝶
Atmel Bootloader（Butterfly 开发板）(SPM)
蝴蝶_mk
Mikrokopter.de 蝴蝶Bootloader (SPM)
蝴蝶
Mikrokopter.de 蝴蝶Bootloader (SPM)
bwmega
BitWizard ftdi atmega 内置编程器（TPI、ISP）

附录 C：编程器名单
83
c232hm
FTDI（TPI、ISP）的 C232HM 电缆
c2n232i
串口：reset=dtr sck=!rts sdo=!txd sdi=!cts（TPI、ISP）
ch341a
CH341A 编程器：注意 AVR F CPU > 6.8 MHz (ISP)
达帕
直接 AVR 并行接入电缆（TPI、ISP）
达萨
串口：reset=rts sck=dtr sdo=txd sdi=cts（TPI、ISP）
达萨3
串口：reset=!dtr sck=rts sdo=txd sdi=cts（TPI、ISP）
Digilent-HS2
Digilent JTAG HS2 (MPSSE)（TPI、ISP）
龙_dw
Atmel AVR Dragon（调试WIRE）
龙_hvsp
爱特梅尔 AVR Dragon (HVSP)
龙ISP
Atmel AVR Dragon（TPI、ISP）
龙_jtag
Atmel AVR Dragon（JTAG、XMEGAJTAG、AVR32JTAG）
龙_pdi
爱特梅尔 AVR Dragon (PDI)
龙_pp
爱特梅尔 AVR Dragon (HVPP)
干靴
无需该部件即可模拟Bootloader编程 (SPM)
试运行
无需编程器即可模拟编程（TPI、ISP、PDI、
UPDI、HVSP、HVPP、aWire）
DT006
Dontronics DT006（TPI、ISP）
埃哈乔ISP
来自 eHaJo.de 的 AVR ISP 编程器（TPI、ISP）
埃雷-isp-avr
ERE ISP-AVR（TPI、ISP）
翻转1
使用 USB DFU v1 的 FLIP Bootloader (doc7618) (SPM)
翻转2
使用 USB DFU v2 (AVR4023) (SPM) 的 FLIP Bootloader
苍蝇拍2
TinCan 工具苍蝇拍 2（TPI、ISP）
弗兰克-STK200
弗兰克 STK200（TPI、ISP）

ft2232h
FT2232H/D 编程器（TPI、ISP）
ft2232h_jtag
基于FT2232H的通用JTAG编程器（JTAG）
英尺232小时
基于FT232H的通用编程器（TPI、ISP）
ft232h_jtag
基于FT232H的通用JTAG编程器（JTAG）
ft232r
基于 FT232R 的通用编程器（TPI、ISP）
ft245r
基于 FT245R 的通用编程器（TPI、ISP）
ft4232h
FT4232H编程器（TPI、ISP）
未来莱克
Futurlec.com 编程电缆（TPI、ISP）
伊塞夫程序
来自 iascaled.com 的 AVR ISP 编程器（TPI、ISP）
jtag1慢速
Atmel JTAG ICE mkI (JTAGmkI)
jtag2dw
Atmel JTAG ICE mkII (debugWIRE)
jtag2fast
Atmel JTAG ICE mkII（JTAG、XMEGAJTAG、AVR32JTAG）
标签2
Atmel JTAG ICE mkII（JTAG、XMEGAJTAG、AVR32JTAG）
jtag2isp
Atmel JTAG ICE mkII（TPI、ISP）
jtag2pdi
Atmel JTAG ICE mkII (PDI)
jtag2慢速
Atmel JTAG ICE mkII（JTAG、XMEGAJTAG、AVR32JTAG）
jtag2updi
JTAGv2 至 UPDI 桥接器 (UPDI)
纳米每个
JTAGv2 至 UPDI 桥接器 (UPDI)
标签3
Atmel AVR JTAGICE3（JTAG、XMEGAJTAG、AVR32JTAG）
jtag3dw
Atmel AVR JTAGICE3（调试WIRE）
jtag3isp
Atmel AVR JTAGICE3 (ISP)
jtag3pdi
爱特梅尔 AVR JTAGICE3 (PDI)
jtag3updi
爱特梅尔 AVR JTAGICE3 (UPDI)

附录 C：编程器名单
84
标签键
Amontec JTAGKey/JTAGKey-Tiny/JTAGKey2（TPI、ISP）
jtagmkI
Atmel JTAG ICE mkI (JTAGmkI)
标签1
Atmel JTAG ICE mkI (JTAGmkI)
jtagmkII
Atmel JTAG ICE mkII（JTAG、XMEGAJTAG、AVR32JTAG）
jtagmkII_avr32
Atmel JTAG ICE mkII (aWire)
jtag2avr32
Atmel JTAG ICE mkII (aWire)
酷特链接
KT-LINK FT2232H：IO 切换、电压缓冲器（TPI、ISP）
linuxspi
在 /dev/spidev* 中使用 Linux SPI 设备（TPI、ISP）
LM3S811
Luminary Micro LM3S811 评估板（修订版 A）（TPI、ISP）
米巴510
Crossbow MIB510编程板（TPI、ISP）
微核
微核Bootloader (SPM)
尼博比
NIBObee（TPI、ISP）
欧联
O-Link、OpenJTAG ARM JTAG USB（TPI、ISP）
开放莫科
Openmoko 调试板 (v3)（TPI、ISP）
帕夫尔
Jason Kyle 的 pAVR 串行编程器 (ISP)
挑选2
Microchip PICkit 2 编程器 (ISP)
皮克特4
MPLAB(R) PICkit 4（JTAG、XMEGAJTAG）
pickit4_jtag
MPLAB(R) PICkit 4（JTAG、XMEGAJTAG）
pickit4_dw
MPLAB(R) PICkit 4（调试WIRE）
pickit4_isp
MPLAB(R) PICkit 4 (ISP)
pickit4_mplab
MPLAB(R) PICkit 4（JTAG、XMEGAJTAG）
pickit4_mplab_jtag
MPLAB(R) PICkit 4（JTAG、XMEGAJTAG）
pickit4_mplab_dw

MPLAB(R) PICkit 4（调试WIRE）
pickit4_mplab_isp
MPLAB(R) PICkit 4 (ISP)
pickit4_mplab_pdi
MPLAB(R) PICkit 4 (PDI)
pickit4_mplab_tpi
MPLAB(R) PICkit 4 (TPI)
pickit4_mplab_updi
MPLAB(R) PICkit 4 (UPDI)
pickit4_pdi
MPLAB(R) PICkit 4 (PDI)
pickit4_tpi
MPLAB(R) PICkit 4 (TPI)
pickit4_updi
MPLAB(R) PICkit 4 (UPDI)
挑选5
MPLAB(R) PICkit 5（JTAG、XMEGAJTAG）
pickit5_jtag
MPLAB(R) PICkit 5（JTAG、XMEGAJTAG）
pickit5_dw
MPLAB(R) PICkit 5（调试WIRE）
pickit5_isp
MPLAB(R) PICkit 5 (ISP)
pickit5_pdi
MPLAB(R) PICkit 5 (PDI)
pickit5_tpi
MPLAB(R) PICkit 5 (TPI)
pickit5_updi
MPLAB(R) PICkit 5 (UPDI)
pickit_basic
MPLAB(R) PICkit 基本版（JTAG、XMEGAJTAG）
pickit_basic_mplab
MPLAB(R) PICkit 基本版（JTAG、XMEGAJTAG）
pickit_basic_jtag
MPLAB(R) PICkit 基本版（JTAG、XMEGAJTAG）
pickit_basic_
mplab_jtag
MPLAB(R) PICkit 基本版（JTAG、XMEGAJTAG）
pickit_basic_dw
MPLAB(R) PICkit Basic（debugWIRE）
pickit_basic_
mplab_dw
MPLAB(R) PICkit Basic（debugWIRE）
pickit_basic_isp
MPLAB(R) PICkit 基本版 (ISP)

附录 C：编程器名单
85
pickit_basic_
mplab_isp
MPLAB(R) PICkit 基本版 (ISP)
pickit_basic_pdi
MPLAB(R) PICkit 基本版 (PDI)
pickit_basic_
mplab_pdi
MPLAB(R) PICkit 基本版 (PDI)
pickit_basic_tpi
MPLAB(R) PICkit 基本版 (TPI)
pickit_basic_
mplab_tpi
MPLAB(R) PICkit 基本版 (TPI)
pickit_basic_updi
MPLAB(R) PICkit 基本版 (UPDI)
pickit_basic_
mplab_updi
MPLAB(R) PICkit 基本版 (UPDI)
微微网
Picoweb 编程电缆（TPI、ISP）
pkobn_updi
好奇号纳米 (nEDBG) (UPDI)
小马-STK200
Pony Prog STK200（TPI、ISP）
波尼瑟
ponyprog 串行：reset=!txd sck=rts sdo=dtr sdi=cts（TPI、ISP）
电源调试器
Atmel PowerDebugger（JTAG、XMEGAJTAG、AVR32JTAG）
powerdebugger_jtag
Atmel PowerDebugger（JTAG、XMEGAJTAG、AVR32JTAG）
powerdebugger_dw
Atmel PowerDebugger (debugWIRE)
powerdebugger_isp
Atmel 电源调试器 (ISP)
powerdebugger_pdi
Atmel 电源调试器 (PDI)
powerdebugger_tpi
Atmel 电源调试器 (TPI)
powerdebugger_updi
Atmel 电源调试器 (UPDI)
树莓派_GPIO
通过 sysfs/libgpiod (ISP) 的 Raspberry Pi GPIO
序列号
串行UPDI (UPDI)
塞尔普罗格

通过 Flashrom (ISP) 的 Serprog 协议进行编程
西普罗格
Lancos SI-Prog（与ponyser相同）（TPI、ISP）
快照
MPLAB(R) SNAP（JTAG、XMEGAJTAG）
快照_jtag
MPLAB(R) SNAP（JTAG、XMEGAJTAG）
快照_dw
MPLAB(R) SNAP（调试WIRE）
快照ISP
MPLAB(R) SNAP (ISP)
snap_mplab
MPLAB(R) SNAP（JTAG、XMEGAJTAG）
snap_mplab_jtag
MPLAB(R) SNAP（JTAG、XMEGAJTAG）
snap_mplab_dw
MPLAB(R) SNAP（调试WIRE）
snap_mplab_isp
MPLAB(R) SNAP (ISP)
snap_mplab_pdi
MPLAB(R) SNAP (PDI)
snap_mplab_tpi
MPLAB(R) SNAP (TPI)
snap_mplab_updi
MPLAB(R) SNAP (UPDI)
快照_pdi
MPLAB(R) SNAP (PDI)
快照_tpi
MPLAB(R) SNAP (TPI)
快照updi
MPLAB(R) SNAP (UPDI)
SP12
Steve Bolt 的编程器（TPI、ISP）
STK200
STK200 入门套件（TPI、ISP）
STK500
Atmel STK500（先探测 v2，然后探测 v1）(ISP)
stk500hvsp
爱特梅尔 STK500 v2 (HVSP)
scrapmonkey_hvsp
爱特梅尔 STK500 v2 (HVSP)
STK500PP
爱特梅尔 STK500 v2 (HVPP)

附录 C：编程器名单
86
scrapmonkey_pp
爱特梅尔 STK500 v2 (HVPP)
STK500V1
爱特梅尔 STK500 v1 (ISP)
STK500V2
爱特梅尔 STK500 v2（TPI、ISP）
刮毛猴
爱特梅尔 STK500 v2（TPI、ISP）
STK600
爱特梅尔 STK600（ISP、PDI）
STK600HVSP
爱特梅尔 STK600 (HVSP)
STK600PP
爱特梅尔 STK600 (HVPP)
TC2030
Tag-Connect TC2030（TPI、ISP）
很小的
Teensy Bootloader (SPM)
泰格德
Tigard 接口板（TPI、ISP）
TTL232R
FTDI TTL232R-5V，带 ICSP 适配器（TPI、ISP）
通帕
TIAO USB 多协议适配器（TPI、ISP）
通帕-b
TIAO USB 多协议适配器（TPI、ISP）
tumpa_jtag
TIAO USB 多协议适配器 (JTAG)
嗯232h
FTDI（TPI、ISP）的 UM232H 模块
不兼容
与短接的所有引脚对不兼容（TPI、ISP）
你的时钟
使用 urprotocol (SPM) 的 Urboot Bootloader
usbasp
USBasp ISP 和 TPI 编程器（TPI、ISP）
usbasp克隆
任何具有正确 VID/PID（TPI、ISP）的 usbasp 克隆
微型USB
USBtiny 简单 USB 编程器（TPI、ISP）
接线
使用 STK500 v2 协议 (SPM) 连接Bootloader
蜜蜂
XBeeBoot 无线Bootloader (STK500 v1) (SPM)
西里

Xilinx JTAG 电缆（TPI、ISP）
xplained迷你
Atmel XplainedMini (ISP)
xplainedmini_isp
Atmel XplainedMini (ISP)
xplainedmini_dw
Atmel XplainedMini（调试WIRE）
xplainedmini_tpi
爱特梅尔 XplainedMini (TPI)
xplainedmini_updi
爱特梅尔 XplainedMini (UPDI)
xplainedpro
Atmel XplainedPro（JTAG、XMEGAJTAG、AVR32JTAG）
xplainedpro_jtag
Atmel XplainedPro（JTAG、XMEGAJTAG、AVR32JTAG）
xplainedpro_pdi
爱特梅尔 XplainedPro (PDI)
xplainedpro_updi
爱特梅尔 XplainedPro (UPDI)

87
附录 D 零件清单
AVRDUDE 支持以下部件：左栏列出部件的 id，右栏列出部件的 id
列包含其正式部件名称；替代名称（如果有）；以及可用的列表
括号内为编程接口；请参见第 4.2 节 [编程器定义]，第 54 页）。
AVRDUDE 配置文件中包含有关每个部分的更多详细信息。
uc3a0512
AT32UC3A0512、AT32UC3A0512AU（AVR32JTAG、aWire）
89S51
AT89S51（ISP、HVPP）
89S52
AT89S52（ISP、HVPP）
c128
AT90CAN128（SPM、ISP、HVPP、JTAG、JTAGmkI）
c32
AT90CAN32（SPM、ISP、HVPP、JTAG）
c64
AT90CAN64（SPM、ISP、HVPP、JTAG）
脉宽调制1
AT90PWM1（SPM、ISP、HVPP、debugWIRE）
脉宽调制161
AT90PWM161（SPM、ISP、HVPP、debugWIRE）
脉宽调制2
AT90PWM2（SPM、ISP、HVPP、debugWIRE）
脉宽调制216
AT90PWM216（SPM、ISP、HVPP、debugWIRE）
脉宽调制2b
AT90PWM2B（SPM、ISP、HVPP、debugWIRE）
脉宽调制3
AT90PWM3（SPM、ISP、HVPP、debugWIRE）
脉宽调制316
AT90PWM316（SPM、ISP、HVPP、debugWIRE）
脉宽调制3b
AT90PWM3B（SPM、ISP、HVPP、debugWIRE）
脉宽调制81
AT90PWM81、AT90PWM81EP（SPM、ISP、HVPP、debugWIRE）
1200
AT90S1200、AT90S1200A（SPM、ISP、HVPP）
2313
AT90S2313（SPM、ISP、HVPP）
2323
AT90S2323（SPM、ISP、HVSP）
2333
AT90S2333（SPM、ISP、HVPP）
2343
AT90S2343（SPM、ISP、HVSP）
4414

AT90S4414（SPM、ISP、HVPP）
4433
AT90S4433（SPM、ISP、HVPP）
4434
AT90S4434（SPM、ISP、HVPP）
8515
AT90S8515（SPM、ISP、HVPP）
8535
AT90S8535（SPM、ISP、HVPP）
USB1286
AT90USB1286（SPM、ISP、HVPP、JTAG）
USB1287
AT90USB1287（SPM、ISP、HVPP、JTAG）
USB162
AT90USB162（SPM、ISP、HVPP、debugWIRE）
USB646
AT90USB646（SPM、ISP、HVPP、JTAG）
USB647
AT90USB647（SPM、ISP、HVPP、JTAG）
USB82
AT90USB82（SPM、ISP、HVPP、debugWIRE）
a5505
ATA5505（SPM、ISP、HVPP、debugWIRE）
a6612c
ATA6612C（SPM、ISP、HVPP、debugWIRE）
a6613c
ATA6613C（SPM、ISP、HVPP、debugWIRE）
a6614q
ATA6614Q（SPM、ISP、HVPP、debugWIRE）
a6616c
ATA6616C（SPM、ISP、HVPP、debugWIRE）
a6617c
ATA6617C（SPM、ISP、HVPP、debugWIRE）
a664251
ATA664251（SPM、ISP、HVPP、debugWIRE）
m103
ATmega103、ATmega103L（SPM、ISP、HVPP）
m128
ATmega128、ATmega128L（SPM、ISP、HVPP、JTAG、JTAGmkI）

附录 D：零件清单
88
米1280
ATmega1280、ATmega1280V（SPM、ISP、HVPP、JTAG）
m1281
ATmega1281、ATmega1281V（SPM、ISP、HVPP、JTAG）
m1284
ATmega1284（SPM、ISP、HVPP、JTAG）
m1284p
ATmega1284P（SPM、ISP、HVPP、JTAG）
m1284rfr2
ATmega1284RFR2（SPM、ISP、HVPP、JTAG）
m128a
ATmega128A（SPM、ISP、HVPP、JTAG、JTAGmkI）
m128rfa1
ATmega128RFA1（SPM、ISP、HVPP、JTAG）
m128rfr2
ATmega128RFR2（SPM、ISP、HVPP、JTAG）
米16
ATmega16、ATmega16L（SPM、ISP、HVPP、JTAG、JTAGmkI）
米1608
ATmega1608（SPM、UPDI）
m1609
ATmega1609（SPM、UPDI）
m161
ATmega161、ATmega161L（SPM、ISP、HVPP）
m162
ATmega162、ATmega162L、ATmega162V（SPM、ISP、HVPP、JTAG、
JTAGmkI)
m163
ATmega163、ATmega163L（SPM、ISP、HVPP）
m164a
ATmega164A（SPM、ISP、HVPP、JTAG）
m164p
ATmega164P、ATmega164PV（SPM、ISP、HVPP、JTAG）
m164pa
ATmega164PA（SPM、ISP、HVPP、JTAG）
m165
ATmega165、ATmega165V（SPM、ISP、HVPP、JTAG）
m165a
ATmega165A（SPM、ISP、HVPP、JTAG）
m165p
ATmega165P、ATmega165PV（SPM、ISP、HVPP、JTAG）
m165pa
ATmega165PA（SPM、ISP、HVPP、JTAG）
m168
ATmega168、ATmega168V（SPM、ISP、HVPP、debugWIRE）

m168a
ATmega168A（SPM、ISP、HVPP、debugWIRE）
m168p
ATmega168P、ATmega168PV（SPM、ISP、HVPP、debugWIRE）
m168pa
ATmega168PA（SPM、ISP、HVPP、debugWIRE）
m168pb
ATmega168PB（SPM、ISP、HVPP、debugWIRE）
m169
ATmega169、ATmega169L、ATmega169V（SPM、ISP、HVPP、JTAG、JTAGmkI）
m169a
ATmega169A（SPM、ISP、HVPP、JTAG、JTAGmkI）
m169p
ATmega169P、ATmega169PV（SPM、ISP、HVPP、JTAG）
m169pa
ATmega169PA（SPM、ISP、HVPP、JTAG）
m16a
ATmega16A（SPM、ISP、HVPP、JTAG、JTAGmkI）
m16hva
ATmega16HVA（SPM、ISP、HVSP、debugWIRE）
m16hvb
ATmega16HVB（SPM、ISP、HVPP、debugWIRE）
m16hvbrevb
ATmega16HVBrevB、ATMEGA16HVB（SPM、ISP、HVPP、调试-电线）
米16米1
ATmega16M1（SPM、ISP、HVPP、debugWIRE）
m16u2
ATmega16U2（SPM、ISP、HVPP、debugWIRE）
m16u4
ATmega16U4、ATmega16U4RC（SPM、ISP、HVPP、JTAG）
米2560
ATmega2560、ATmega2560V（SPM、ISP、HVPP、JTAG）
m2561
ATmega2561、ATmega2561V（SPM、ISP、HVPP、JTAG）
m2564rfr2
ATmega2564RFR2（SPM、ISP、HVPP、JTAG）
m256rfr2
ATmega256RFR2（SPM、ISP、HVPP、JTAG）
米32
ATmega32、ATmega32L（SPM、ISP、HVPP、JTAG、JTAGmkI）
m3208
ATmega3208（SPM、UPDI）

附录 D：零件清单
89
m3209
ATmega3209（SPM、UPDI）
m324a
ATmega324A（SPM、ISP、HVPP、JTAG）
m324p
ATmega324P、ATmega324PV（SPM、ISP、HVPP、JTAG）
m324pa
ATmega324PA（SPM、ISP、HVPP、JTAG）
m324pb
ATmega324PB（SPM、ISP、HVPP、JTAG）
米325
ATmega325、ATmega325V（SPM、ISP、HVPP、JTAG）
米3250
ATmega3250、ATmega3250V（SPM、ISP、HVPP、JTAG）
m3250a
ATmega3250A（SPM、ISP、HVPP、JTAG）
m3250p
ATmega3250P、ATmega3250PV（SPM、ISP、HVPP、JTAG）
m3250pa
ATmega3250PA（SPM、ISP、HVPP、JTAG）
m325a
ATmega325A（SPM、ISP、HVPP、JTAG）
m325p
ATmega325P、ATmega325PV（SPM、ISP、HVPP、JTAG）
米325帕
ATmega325PA（SPM、ISP、HVPP、JTAG）
m328
ATmega328（SPM、ISP、HVPP、debugWIRE）
m328p
ATmega328P（SPM、ISP、HVPP、debugWIRE）
m328pb
ATmega328PB（SPM、ISP、HVPP、debugWIRE）
m329
ATmega329、ATmega329V（SPM、ISP、HVPP、JTAG）
m3290
ATmega3290、ATmega3290V（SPM、ISP、HVPP、JTAG）
m3290a
ATmega3290A（SPM、ISP、HVPP、JTAG）
m3290p
ATmega3290P、ATmega3290PV（SPM、ISP、HVPP、JTAG）
m3290pa
ATmega3290PA（SPM、ISP、HVPP、JTAG）
m329a
ATmega329A（SPM、ISP、HVPP、JTAG）
m329p

ATmega329P、ATmega329PV（SPM、ISP、HVPP、JTAG）
m329pa
ATmega329PA（SPM、ISP、HVPP、JTAG）
米32a
ATmega32A（SPM、ISP、HVPP、JTAG、JTAGmkI）
m32c1
ATmega32C1（SPM、ISP、HVPP、debugWIRE）
m32hvb
ATmega32HVB（SPM、ISP、HVPP、debugWIRE）
m32hvbrevb
ATmega32HVBrevB、ATMEGA32HVB（SPM、ISP、HVPP、调试-
电线）
m32hve2
ATmega32HVE2（SPM、ISP、HVSP、debugWIRE）
米32米1
ATmega32M1（SPM、ISP、HVPP、debugWIRE）
m32u2
ATmega32U2（SPM、ISP、HVPP、debugWIRE）
m32u4
ATmega32U4、ATmega32U4RC（SPM、ISP、HVPP、JTAG）
m406
ATmega406（SPM、HVPP、JTAG）
m48
ATmega48、ATmega48V（SPM、ISP、HVPP、debugWIRE）
m4808
ATmega4808（SPM、UPDI）
m4809
ATmega4809（SPM、UPDI）
m48a
ATmega48A（SPM、ISP、HVPP、debugWIRE）
m48p
ATmega48P、ATmega48PV（SPM、ISP、HVPP、debugWIRE）
米48帕
ATmega48PA（SPM、ISP、HVPP、debugWIRE）
m48pb
ATmega48PB（SPM、ISP、HVPP、debugWIRE）
m64
ATmega64、ATmega64L（SPM、ISP、HVPP、JTAG、JTAGmkI）
米640
ATmega640、ATmega640V（SPM、ISP、HVPP、JTAG）
米644
ATmega644、ATmega644V（SPM、ISP、HVPP、JTAG）
m644a
ATmega644A（SPM、ISP、HVPP、JTAG）
m644p
ATmega644P、ATmega644PV（SPM、ISP、HVPP、JTAG）

附录 D：零件清单
90
m644pa
ATmega644PA（SPM、ISP、HVPP、JTAG）
m644rfr2
ATmega644RFR2（SPM、ISP、HVPP、JTAG）
米645
ATmega645、ATmega645V（SPM、ISP、HVPP、JTAG）
米6450
ATmega6450、ATmega6450V（SPM、ISP、HVPP、JTAG）
m6450a
ATmega6450A（SPM、ISP、HVPP、JTAG）
m6450p
ATmega6450P（SPM、ISP、HVPP、JTAG）
米645a
ATmega645A（SPM、ISP、HVPP、JTAG）
m645p
ATmega645P（SPM、ISP、HVPP、JTAG）
米649
ATmega649、ATmega649V（SPM、ISP、HVPP、JTAG）
米6490
ATmega6490、ATmega6490V（SPM、ISP、HVPP、JTAG）
m6490a
ATmega6490A（SPM、ISP、HVPP、JTAG）
m6490p
ATmega6490P（SPM、ISP、HVPP、JTAG）
m649a
ATmega649A（SPM、ISP、HVPP、JTAG）
m649p
ATmega649P（SPM、ISP、HVPP、JTAG）
m64a
ATmega64A（SPM、ISP、HVPP、JTAG、JTAGmkI）
m64c1
ATmega64C1（SPM、ISP、HVPP、debugWIRE）
m64hve2
ATmega64HVE2（SPM、ISP、HVSP、debugWIRE）
米64米1
ATmega64M1（SPM、ISP、HVPP、debugWIRE）
m64rfr2
ATmega64RFR2（SPM、ISP、HVPP、JTAG）
米8
ATmega8、ATmega8L（SPM、ISP、HVPP）
米808
ATmega808（SPM、UPDI）
米809
ATmega809（SPM、UPDI）
m8515

ATmega8515、ATmega8515L（SPM、ISP、HVPP）
m8535
ATmega8535、ATmega8535L（SPM、ISP、HVPP）
米88
ATmega88、ATmega88V（SPM、ISP、HVPP、debugWIRE）
m88a
ATmega88A（SPM、ISP、HVPP、debugWIRE）
m88p
ATmega88P、ATmega88PV（SPM、ISP、HVPP、debugWIRE）
m88pa
ATmega88PA（SPM、ISP、HVPP、debugWIRE）
m88pb
ATmega88PB（SPM、ISP、HVPP、debugWIRE）
m8a
ATmega8A（SPM、ISP、HVPP）
m8hva
ATmega8HVA（SPM、ISP、HVSP、debugWIRE）
m8u2
ATmega8U2（SPM、ISP、HVPP、debugWIRE）
女士128
ATmegaS128（SPM、ISP、HVPP、JTAG、JTAGmkI）
MS64M1
ATmegaS64M1（SPM、ISP、HVPP、debugWIRE）
t10
ATtiny10 (TPI)
t102
ATtiny102、ATtiny102F (TPI)
t104
ATtiny104、ATtiny104F (TPI)
t11
ATtiny11、ATtiny11L (HVSP)
t12
ATtiny12、ATtiny12L、ATtiny12V（ISP、HVSP）
t13
ATtiny13、ATtiny13V（SPM、ISP、HVSP、debugWIRE）
t13a
ATtiny13A（SPM、ISP、HVSP、debugWIRE）
t15
ATtiny15、ATtiny15L（ISP、HVSP）
t1604
ATtiny1604（SPM、UPDI）
t1606
ATtiny1606（SPM、UPDI）
t1607
ATtiny1607（SPM、UPDI）
t1614
ATtiny1614（SPM、UPDI）
t1616

ATtiny1616（SPM、UPDI）

附录 D：零件清单
91
t1617
ATtiny1617（SPM、UPDI）
t1624
ATtiny1624（SPM、UPDI）
t1626
ATtiny1626（SPM、UPDI）
t1627
ATtiny1627（SPM、UPDI）
t1634
ATtiny1634、ATtiny1634R（SPM、ISP、HVPP、debugWIRE）
t1634r
ATtiny1634R（SPM、ISP、HVPP、debugWIRE）
t167
ATtiny167（SPM、ISP、HVPP、debugWIRE）
t20
ATtiny20 (TPI)
t202
ATtiny202（SPM、UPDI）
t204
ATtiny204（SPM、UPDI）
t212
ATtiny212（SPM、UPDI）
t214
ATtiny214（SPM、UPDI）
t22
ATtiny22、ATtiny22L（SPM、ISP、HVSP）
t2313
ATtiny2313、ATtiny2313V（SPM、ISP、HVPP、debugWIRE）
t2313a
ATtiny2313A（SPM、ISP、HVPP、debugWIRE）
t24
ATtiny24、ATtiny24V（SPM、ISP、HVSP、debugWIRE）
t24a
ATtiny24A（SPM、ISP、HVSP、debugWIRE）
t25
ATtiny25、ATtiny25V（SPM、ISP、HVSP、debugWIRE）
t26
ATtiny26、ATtiny26L（ISP、HVPP）
t261
ATtiny261、ATtiny261V（SPM、ISP、HVPP、debugWIRE）
t261a
ATtiny261A（SPM、ISP、HVPP、debugWIRE）
t28
ATtiny28、ATtiny28L、ATtiny28V (HVPP)
t3216

ATtiny3216（SPM、UPDI）
t3217
ATtiny3217（SPM、UPDI）
t3224
ATtiny3224（SPM、UPDI）
t3226
ATtiny3226（SPM、UPDI）
t3227
ATtiny3227（SPM、UPDI）
t4
ATtiny4 (TPI)
t40
ATtiny40 (TPI)
t402
ATtiny402（SPM、UPDI）
t404
ATtiny404（SPM、UPDI）
t406
ATtiny406（SPM、UPDI）
t412
ATtiny412（SPM、UPDI）
t414
ATtiny414（SPM、UPDI）
t416
ATtiny416（SPM、UPDI）
t416汽车
ATtiny416auto、ATtiny416（SPM、UPDI）
t417
ATtiny417（SPM、UPDI）
t424
ATtiny424（SPM、UPDI）
t426
ATtiny426（SPM、UPDI）
t427
ATtiny427（SPM、UPDI）
t4313
ATtiny4313（SPM、ISP、HVPP、debugWIRE）
t43u
ATtiny43U（SPM、ISP、HVPP、debugWIRE）
t44
ATtiny44、ATtiny44V（SPM、ISP、HVSP、debugWIRE）
t441
ATtiny441（SPM、ISP、HVSP、debugWIRE）
t44a
ATtiny44A（SPM、ISP、HVSP、debugWIRE）
t45
ATtiny45、ATtiny45V（SPM、ISP、HVSP、debugWIRE）
t461

ATtiny461、ATtiny461V（SPM、ISP、HVPP、debugWIRE）

附录 D：零件清单
92
t461a
ATtiny461A（SPM、ISP、HVPP、debugWIRE）
t48
ATtiny48（SPM、ISP、HVPP、debugWIRE）
t5
ATtiny5 (TPI)
t804
ATtiny804（SPM、UPDI）
t806
ATtiny806（SPM、UPDI）
t807
ATtiny807（SPM、UPDI）
t814
ATtiny814（SPM、UPDI）
t816
ATtiny816（SPM、UPDI）
t817
ATtiny817（SPM、UPDI）
t824
ATtiny824（SPM、UPDI）
t826
ATtiny826（SPM、UPDI）
t827
ATtiny827（SPM、UPDI）
t828
ATtiny828、ATtiny828R（SPM、ISP、HVPP、debugWIRE）
t828r
ATtiny828R（SPM、ISP、HVPP、debugWIRE）
t84
ATtiny84、ATtiny84V（SPM、ISP、HVSP、debugWIRE）
t841
ATtiny841（SPM、ISP、HVSP、debugWIRE）
t84a
ATtiny84A（SPM、ISP、HVSP、debugWIRE）
t85
ATtiny85、ATtiny85V（SPM、ISP、HVSP、debugWIRE）
t861
ATtiny861、ATtiny861V（SPM、ISP、HVPP、debugWIRE）
t861a
ATtiny861A（SPM、ISP、HVPP、debugWIRE）
t87
ATtiny87（SPM、ISP、HVPP、debugWIRE）
t88
ATtiny88（SPM、ISP、HVPP、debugWIRE）
t9

ATtiny9 (TPI)
x128a1
ATxmega128A1（SPM、PDI、XMEGAJTAG）
x128a1d
ATxmega128A1revD（SPM、PDI、XMEGAJTAG）
x128a1u
ATxmega128A1U（SPM、PDI、XMEGAJTAG）
x128a3
ATxmega128A3（SPM、PDI、XMEGAJTAG）
x128a3u
ATxmega128A3U（SPM、PDI、XMEGAJTAG）
x128a4
ATxmega128A4（SPM、PDI、XMEGAJTAG）
x128a4u
ATxmega128A4U（SPM、PDI）
x128b1
ATxmega128B1（SPM、PDI、XMEGAJTAG）
x128b3
ATxmega128B3（SPM、PDI、XMEGAJTAG）
x128c3
ATxmega128C3（SPM、PDI）
x128d3
ATxmega128D3（SPM、PDI）
x128d4
ATxmega128D4（SPM、PDI）
x16a4
ATxmega16A4（SPM、PDI）
x16a4u
ATxmega16A4U（SPM、PDI）
x16c4
ATxmega16C4（SPM、PDI）
x16d4
ATxmega16D4（SPM、PDI）
x16e5
ATxmega16E5（SPM、PDI）
x192a1
ATxmega192A1（SPM、PDI、XMEGAJTAG）
x192a3
ATxmega192A3（SPM、PDI、XMEGAJTAG）
x192a3u
ATxmega192A3U（SPM、PDI、XMEGAJTAG）
x192c3
ATxmega192C3（SPM、PDI）
x192d3
ATxmega192D3（SPM、PDI）
x256a1
ATxmega256A1（SPM、PDI、XMEGAJTAG）
x256a3

ATxmega256A3（SPM、PDI、XMEGAJTAG）

附录 D：零件清单
93
x256a3b
ATxmega256A3B（SPM、PDI、XMEGAJTAG）
x256a3bu
ATxmega256A3BU（SPM、PDI、XMEGAJTAG）
x256a3u
ATxmega256A3U（SPM、PDI、XMEGAJTAG）
x256c3
ATxmega256C3（SPM、PDI）
x256d3
ATxmega256D3（SPM、PDI）
x32a4
ATxmega32A4（SPM、PDI）
x32a4u
ATxmega32A4U（SPM、PDI）
x32c3
ATxmega32C3（SPM、PDI）
x32c4
ATxmega32C4（SPM、PDI）
x32d3
ATxmega32D3（SPM、PDI）
x32d4
ATxmega32D4（SPM、PDI）
x32e5
ATxmega32E5（SPM、PDI）
x384c3
ATxmega384C3（SPM、PDI）
x384d3
ATxmega384D3（SPM、PDI）
x64a1
ATxmega64A1（SPM、PDI、XMEGAJTAG）
x64a1u
ATxmega64A1U（SPM、PDI、XMEGAJTAG）
x64a3
ATxmega64A3（SPM、PDI、XMEGAJTAG）
x64a3u
ATxmega64A3U（SPM、PDI、XMEGAJTAG）
x64a4
ATxmega64A4（SPM、PDI、XMEGAJTAG）
x64a4u
ATxmega64A4U（SPM、PDI）
x64b1
ATxmega64B1（SPM、PDI、XMEGAJTAG）
x64b3
ATxmega64B3（SPM、PDI、XMEGAJTAG）
x64c3

ATxmega64C3（SPM、PDI）
x64d3
ATxmega64D3（SPM、PDI）
x64d4
ATxmega64D4（SPM、PDI）
x8e5
ATxmega8E5（SPM、PDI）
128da28
AVR128DA28、AVR128DA28T（SPM、UPDI）
128da28s
AVR128DA28S（SPM、UPDI）
128da32
AVR128DA32、AVR128DA32T（SPM、UPDI）
128da32s
AVR128DA32S（SPM、UPDI）
128da48
AVR128DA48、AVR128DA48T（SPM、UPDI）
128da48s
AVR128DA48S（SPM、UPDI）
128da64
AVR128DA64、AVR128DA64T（SPM、UPDI）
128da64s
AVR128DA64S（SPM、UPDI）
128分贝28
AVR128DB28、AVR128DB28T（SPM、UPDI）
128分贝32
AVR128DB32、AVR128DB32T（SPM、UPDI）
128分贝48
AVR128DB48、AVR128DB48T（SPM、UPDI）
128分贝64
AVR128DB64、AVR128DB64T（SPM、UPDI）
16dd14
AVR16DD14（SPM、UPDI）
16dd20
AVR16DD20（SPM、UPDI）
16dd28
AVR16DD28（SPM、UPDI）
16dd32
AVR16DD32（SPM、UPDI）
16du14
AVR16DU14（SPM、UPDI）
16du20
AVR16DU20（SPM、UPDI）
16du28
AVR16DU28（SPM、UPDI）
16du32
AVR16DU32（SPM、UPDI）
16个28

AVR16EA28（SPM、UPDI）

附录 D：零件清单
94
16ea32
AVR16EA32（SPM、UPDI）
16ea48
AVR16EA48（SPM、UPDI）
16eb14
AVR16EB14（SPM、UPDI）
16eb20
AVR16EB20（SPM、UPDI）
16eb28
AVR16EB28（SPM、UPDI）
16eb32
AVR16EB32（SPM、UPDI）
32da28
AVR32DA28、AVR32DA28T（SPM、UPDI）
32da28s
AVR32DA28S（SPM、UPDI）
32da32
AVR32DA32、AVR32DA32T（SPM、UPDI）
32da32s
AVR32DA32S（SPM、UPDI）
32da48
AVR32DA48、AVR32DA48T（SPM、UPDI）
32da48s
AVR32DA48S（SPM、UPDI）
32分贝28
AVR32DB28、AVR32DB28T（SPM、UPDI）
32分贝32
AVR32DB32、AVR32DB32T（SPM、UPDI）
32分贝48
AVR32DB48、AVR32DB48T（SPM、UPDI）
32dd14
AVR32DD14（SPM、UPDI）
32dd20
AVR32DD20（SPM、UPDI）
32dd28
AVR32DD28（SPM、UPDI）
32dd32
AVR32DD32（SPM、UPDI）
32du14
AVR32DU14（SPM、UPDI）
32du20
AVR32DU20（SPM、UPDI）
32du28
AVR32DU28（SPM、UPDI）
32度32

AVR32DU32（SPM、UPDI）
32ea28
AVR32EA28（SPM、UPDI）
32个32
AVR32EA32（SPM、UPDI）
32ea48
AVR32EA48（SPM、UPDI）
32eb14
AVR32EB14（SPM、UPDI）
32eb20
AVR32EB20（SPM、UPDI）
32eb28
AVR32EB28（SPM、UPDI）
32eb32
AVR32EB32（SPM、UPDI）
32SD20
AVR32SD20（SPM、UPDI）
32SD28
AVR32SD28（SPM、UPDI）
32SD32
AVR32SD32（SPM、UPDI）
64da28
AVR64DA28、AVR64DA28T（SPM、UPDI）
64da28s
AVR64DA28S（SPM、UPDI）
64da32
AVR64DA32、AVR64DA32T（SPM、UPDI）
64da32s
AVR64DA32S（SPM、UPDI）
64da48
AVR64DA48、AVR64DA48T（SPM、UPDI）
64da48s
AVR64DA48S（SPM、UPDI）
64da64
AVR64DA64、AVR64DA64T（SPM、UPDI）
64da64s
AVR64DA64S（SPM、UPDI）
64分贝28
AVR64DB28、AVR64DB28T（SPM、UPDI）
64分贝32
AVR64DB32、AVR64DB32T（SPM、UPDI）
64分贝48
AVR64DB48、AVR64DB48T（SPM、UPDI）
64分贝64
AVR64DB64、AVR64DB64T（SPM、UPDI）
64dd14
AVR64DD14（SPM、UPDI）
64dd20

AVR64DD20（SPM、UPDI）

附录 D：零件清单
95
64dd28
AVR64DD28（SPM、UPDI）
64dd32
AVR64DD32（SPM、UPDI）
64du28
AVR64DU28（SPM、UPDI）
64du32
AVR64DU32（SPM、UPDI）
64个28
AVR64EA28（SPM、UPDI）
64个32
AVR64EA32（SPM、UPDI）
64ea48
AVR64EA48（SPM、UPDI）
8ea28
AVR8EA28（SPM、UPDI）
8ea32
AVR8EA32（SPM、UPDI）
LGT168P
LGT8F168P（SPM、ISP、HVPP、debugWIRE）
LGT328P
LGT8F328P（SPM、ISP、HVPP、debugWIRE）
LGT88P
LGT8F88P（SPM、ISP、HVPP、debugWIRE）
注释
1. 对 32 位 AVR（通过 aWire 或 AVT32JTAG）的支持充其量只是实验性的。
2. 不过，并非所有编程硬件都支持 128 KB 以上的Flash寻址
大多数人都会支持它。
3. AT90S1200 的 ISP 编程协议与 AT90S1200 的 ISP 编程协议有细微的区别
其他 AVR。因此，并非所有 ISP 编程器都支持该设备。已知的工作都是
直接 bitbang 编程器，以及所有谈论 STK500v2 协议的编程器。
4. 并不是所有的编程器都能为一个部件提供所有的内存。Bootloader永远无法写入
例如，熔丝位。

96
附录 E 存储器列表
E.1 经典部分
经典设备除了 eeprom、flash、signature 之外可能还具有以下存储器
并锁定：
校准
一个或多个字节的 RC 振荡器校准数据
流出物
扩展熔丝字节
熔丝位
仅具有单个熔丝字节的设备中的熔丝字节
熔断器
高熔丝字节
熔断器
低熔丝字节
生产信号
小型只读存储器中的签名、校准字节和序列号，
据记录仅适用于 ATmega324PB、ATmega328PB、
ATtiny102 和 ATtiny104； AVRDUDE 通常会尝试使此内存
可用，也适用于未记录的部分，但不是所有编程器
或许能够读取这段内存
西格罗
prodsig 的内存别名
血清
prodsig的序列号部分；由于文件稀缺，这可能不会
实际上是一个序列号或者可以被一些编程器读取
用户签名
三
额外的
闪光
页面
为了
固件
设置；
这个
存储器
是
不
被抹去
期间
一个
芯片
擦除。
仅
一些
经典
零件,

AT-
mega(64|128|256|644|1284|2564)RFR2，有一个usersig内存。
用户名
与Flash的不同之处在于它不能通过 ISP 访问
串行编程也不由Bootloader写入。 AVRDUDE 提供 JTAG
经典部分用户存储器的编程。与所有Flash类型存储器一样
-U 选项只能写入 0 位，不能写入 1 位。因此，usersig 需要
在将文件写入此内存区域之前被擦除，例如使用 -T
“删除用户名” -U 用户名:w:parameters.hex:i
io
易失性寄存器存储器；它不能通过外部编程方法访问
ods 只能由Bootloader使用，除非Bootloader跳转到
直接应用程序，即无需 WDT 复位
斯拉姆
易失性RAM存储器；与 io 一样，它不能被外部编程访问
E.2 ATxmegas
ATxmega 器件除了 eeprom、flash、signature 之外还有以下存储器
并锁定：
应用
应用Flash区域
应用表
应用表Flash区域

附录 E：内存列表
97
启动
启动Flash区域
校准
4个（ATxmega-A系列）或5个字节（ATxmega-B/C/D/E）的区域，带有振荡
振荡器校准值；这是 prodsig 的子内存
熔丝位
7字节的逻辑内存，包含一个零件的所有fuseX，可以使用
同时对所有熔断器进行编程；请注意，一些熔丝字节将是
保留，不过
熔丝位0
又名jtaguid：某些设备的 JTAG 用户 ID
熔丝位1
看门狗配置
熔丝位6
ATxmega E 系列部件的故障检测动作配置 TC4/5
熔丝位N
ATxmega 器件的其他熔丝字节，其中 N 为 2、4 或 5，用于系统配置
持续时间
生产信号
生产签名行是工厂亲的只读存储器部分
编程数据，例如振荡器或模拟模块的校准值；它
还包含一个序列号，由生产批号、晶圆组成
零件的编号和晶圆坐标
血清
由10个字节组成的部分具有唯一ID的序列号；这些是
上面 prodsig 内存的一部分
西格罗
prodsig 的内存别名
时间感
两字节内存，位于 prodsig 内；它包含一个12位TEM-
温度传感器校准值
用户签名
可用于固件设置的附加Flash页面；这个存储器-
在整片擦除过程中，逻辑不会被擦除
io
易失性寄存器存储器； AVRDUDE 可以读取该内存，但不能写入
使用外部编程
斯拉姆
易失性RAM存储器；无法通过外部编程有效访问
E.3 现代 AVR 零件
现代8位AVR设备除了eeprom、flash之外还有以下存储器：
签名和锁定：
熔丝位0
又名wdtcfg：看门狗配置
熔丝位1

又名bodcfg：掉电检测配置
熔丝位2
又名osccfg：振荡器配置
熔丝位4
又名tcd0cfg（并非所有设备）：定时器计数器类型 D 配置
熔丝位5
又名syscfg0：系统配置0
熔丝位6
又名syscfg1：系统配置1

附录 E：内存列表
98
熔丝位7
又名附加或代码大小：应用程序代码部分的末尾或
代码大小（以 256/512 字节块为单位）
熔丝位8
又名bootend 或 bootsize：引导部分的末尾或块中的引导大小
256/512 字节
弗塞亚
又名pdicfg：配置/锁定 updi 访问；它是唯一由以下组成的熔丝位
两个字节
熔丝位
最多16字节的逻辑存储器，包含零件的所有fuseX，可以
用于同时对所有熔断器进行编程
osc16错误
通常描述 3 V 时 16 MHz 振荡器频率误差的两个字节
和 5V 分别
osc20错误
通常描述 3 V 时 20 MHz 振荡器频率误差的两个字节
和 5V 分别
osccal16
16 MHz 的两个振荡器校准字节
osccal20
20 MHz 的两个振荡器校准字节
生产信号
用于工厂编程数据（例如签名）的只读存储器部分
校准值和序列号
西格罗
prodsig 的内存别名
血清
具有部件唯一 ID 的序列号（10 或 16 字节）
时间感
温度传感器校准值
布特罗
只能由Bootloader代码中的 MCU 访问的额外内存页；
仅当设备解锁时，UDPI 才能读写此内存
用户行
额外的 EEPROM 内存页可用于固件设置；这个
整片擦除期间不会擦除存储器
同胞
特殊系统信息块存储器，包含有关 AVR 系列的信息，
芯片修改等
io
易失性寄存器存储器； AVRDUDE 可以对这个内存进行编程，但是这是
实用性有限，因为写入 io 内存的任何内容都将是未定义的或
重置后丢失；仍然可以使用写入终端中的各个寄存器，
例如，测试 I/O 端口

斯拉姆
易失性RAM存储器；可以读写，但写入后内容会丢失
重置

99
概念索引
!
! （子外壳）。。。

-b 波特率。。。

。。。。。。。。。12
-t。。。。。。。。。

。。26
-x serprog。。。

概念索引

100
1
1200。。。

16ea32。。。

32dd28。。。

64da32s。。。

。 。 。 。 。 。 。 95
8ea32。 。 。 。 。 。 。

概念索引
101
一个
a5505。。。

arduinoisporg。。。

AT90S8515。。。

ATmega1609。。。

。 。 88
ATmega3209。 。 。

概念索引

102
ATmega324A。。。

ATmega32M1。。。

ATmega64A。。。

Atmel XplainedPro 3, 86
爱特梅尔-ICE. 3, 82
阿特梅利切. 82
atmelice_dw. 82
atmelice_isp. 82
atmelice_jtag 82
atmelice_pdi. 82
atmelice_tpi. 82
atmelice_updi 82
ATtiny10 90
ATtiny102、ATtiny102F. 90
ATtiny104、ATtiny104F. 90
ATtiny11，ATtiny11L. 90
ATtiny12、ATtiny12L、ATtiny12V. 90
ATtiny13，ATtiny13V. 90
ATtiny13A. 90
ATtiny15，ATtiny15L. 90
ATtiny1604. 90
ATtiny1606. 90
ATtiny1607. 90
ATtiny1614. 90
ATtiny1616. 90
ATtiny1617. 90
ATtiny1624. 91

概念索引
103

ATtiny1626。。。

。。。。91
ATtiny402。。。。

ATtiny817。。。

ATxmega16C4。。。

ATxmega64A4 93
ATxmega64A4U 93
ATxmega64B1 93
ATxmega64B3 93
ATxmega64C3 93
ATxmega64D3 93

概念索引

104
ATxmega64D4。。。

。。93
AVR16DU28。。。

AVR32EA32。。。

AVR64EA28。。。

布莱恩・S・迪恩 (Brian S. Dean) 的编程器。。。

概念索引

105
BSD。。。

达萨3。。。

ehajo-isp。。。

ft232h_jtag。。。

概念索引
106
熔丝位1. 。。。

。。。。。。。84
jtag1慢。。。。。。。

。 。 。 。 。 。 84
L
Lancos SI-Prog。 。 。 。 。 。

m1608。。。

概念索引
107
m169p。。。

。。。。。。89
m3250p。。。。。。

m406 89
m48 89
m4808 89
m4809 89
m48a 89
m48p 89
m48pa 89
m48pb 89
m64 89
m640 89
m644 89
m644a 89
m644p 89
m644pa 89
m644rfr2 90
m645 90
m6450 90
m6450a 90
m6450p 90
m645a 90
m645p 90
m649 90
m6490 90
m6490a 90

m6490p。。。

。。。。。。。84
Microchip PICkit 2 编程器。。。。。。。。。。。。。26, 84
微核。。。。。。。。。。。。。

概念索引
108
氧
o 链接。。。

。。。。8
选项-e。。。。

选项 -x jtag2updi。。。

概念索引
109
磷
父部分。。。

pickit4_isp。。。

。。。。。。54
PM_JTAG。。。。。。

。。。。。。。。。。87
脉宽调制3。。。。。。。。。。

概念索引
110
S
节省内存 {addr len} file[:format] 。。。。。。。。。。。。41
sck期。。。。。。。。。。。。

snap_mplab_tpi。。。

。。55
syscfg0。。。

。 。 。 。 。 。 。 91
t204。 。 。 。 。 。 。

t414。。。

概念索引

111

t44a。。。

。。。。。。92
t87。。。。。。

Unix 配置文件。。。

向量表。。。

概念索引
112
X
x128a1。。。

x256a1。。。

x64c3。。。