在写入闪存时响应外设中断

TB3344

简介

作者：Henrik M. Arnesen, Egil Rotevatn，Microchip Technology Inc.

Microchip AVR® EA 系列单片机配备了闪存，其中闪存被划分为可读写同时进行（RWW）区和不可读写同时进行（NRWW）区。这种设计使得 CPU 在对 RWW 区进行擦除或写入操作时，仍可从 NRWW 区继续执行指令。由于对闪存的擦除/写入操作通常需要以毫秒为单位计时，RWW 功能让用户能够充分利用这段时间，无需等待操作完成。本文档旨在介绍如何利用 AVR EA 器件的 NRWW 闪存区对 RWW 闪存区进行写入，以及这样做的优势。

本技术简报将解释 RWW/NRWW 的概念及其在 AVR EA 系列单片机上的实现，并结合以下应用场景进行说明：

在写入闪存时响应外设中断

- 本示例展示了能够从 NRWW 闪存区写入 RWW 区的优势
- 通过不断切换不同器件引脚，显示擦除/写入周期中的重要事件
- 这些引脚分别代表周期性中断、页缓冲区状态和闪存状态
- 在 NRWW 或 RWW 区域擦除和写入程序页时，各事件的发生情况有所不同

注意：针对本文档描述的应用场景，提供了两个代码示例：一个是在 MPLAB® 中使用 Microchip 代码配置器（MCC） 创建的，另一个是在 AVR® EA 上使用 Microchip Studio 开发的裸机示例。两者均可在 Microchip Discover 上获取。

- Microchip Studio代码示例
- MPLAB X代码示例

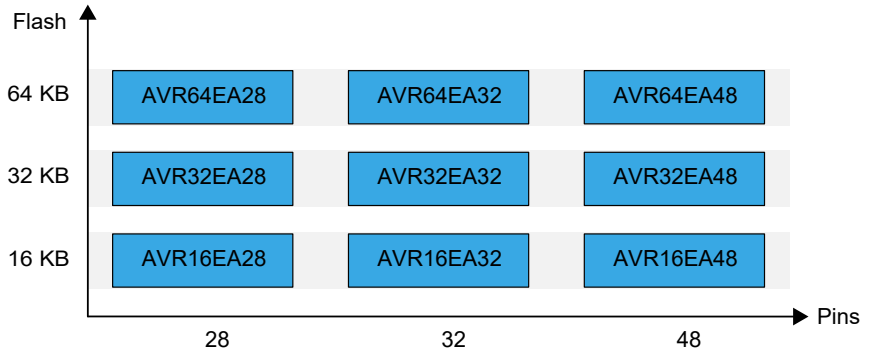
目录

1. 相关器件

本节列出了与本文档相关的器件。下列图示展示了不同系列器件的引脚数量和存储容量的变化：

纵向向上迁移无需修改代码，因为这些器件引脚兼容，并且功能相同或更丰富
横向向左迁移会减少引脚数量，因此可用功能也会减少
不同闪存容量的器件通常也配备不同的 SRAM 和 EEPROM

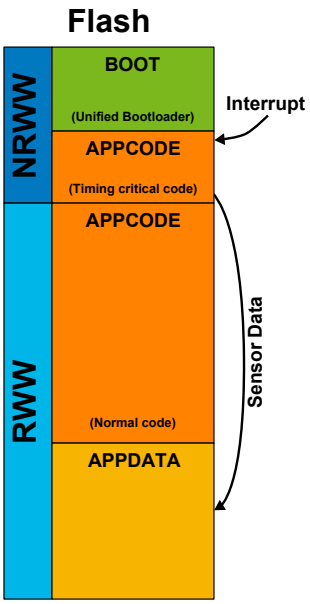
图 1-1. AVR® EA系列概览



2.概述

闪存由若干页组成，每页包含多个字节，在 AVR 架构中一个字为2字节（16位）。以 AVR EA 为例，其每页大小为128字节。无论页大小如何，闪存都被物理划分为两个区域：不可读写同时进行（NRWW）区和可读写同时进行（RWW）区。NRWW 区从闪存起始位置开始，可能会与 BOOT 区和 APPCODE/APPDATA 区重叠，具体取决于熔丝位设置。需要注意的是，无法在代码正在运行的区域同时进行写操作。如果 NRWW 区跨越了 BOOT/APPCODE 或 BOOT/APPDATA 的边界，则必须将不中断运行的代码放在 BOOT 区，以便在 APPCODE/APPDATA 区进行擦除/写入时，代码能够持续运行。如果配置为 BOOT/APPCODE/APPDATA 分区，也可以将不中断的代码放在 APPCODE 区的 NRWW 部分，从而在编程 APPDATA 时无需暂停 CPU。

图 2-1. RWW存储器流程图



“RWW区”这一术语指的是正在被编程（擦除或写入）的区域，而不是正在被读取的区域。只有位于 NRWW 闪存区的代码，才能在对 RWW 区进行编程时通过执行 CPU 指令或读取数据来访问。

由此可见，NRWW 区和 RWW 区的主要区别如下：

1. 当使用位于 NRWW 区的代码对 RWW 区内的某一页进行擦除或写入时，代码和数据仍可在 NRWW 区持续运行和读取，从而实现程序存储器操作期间 CPU 的连续运行。
2. 当对 NRWW 区内的某一页进行擦除或写入时，CPU 会在操作期间暂停。唯一的例外是：当从 BOOT 区向与 NRWW 区重叠的 APPCODE 或 APPDATA 区写入时，CPU 可以不中断运行。

利用 AVR EA 的 NRWW/RWW 程序存储器分区，应用程序可以在写入程序存储器时避免 CPU 被阻塞。

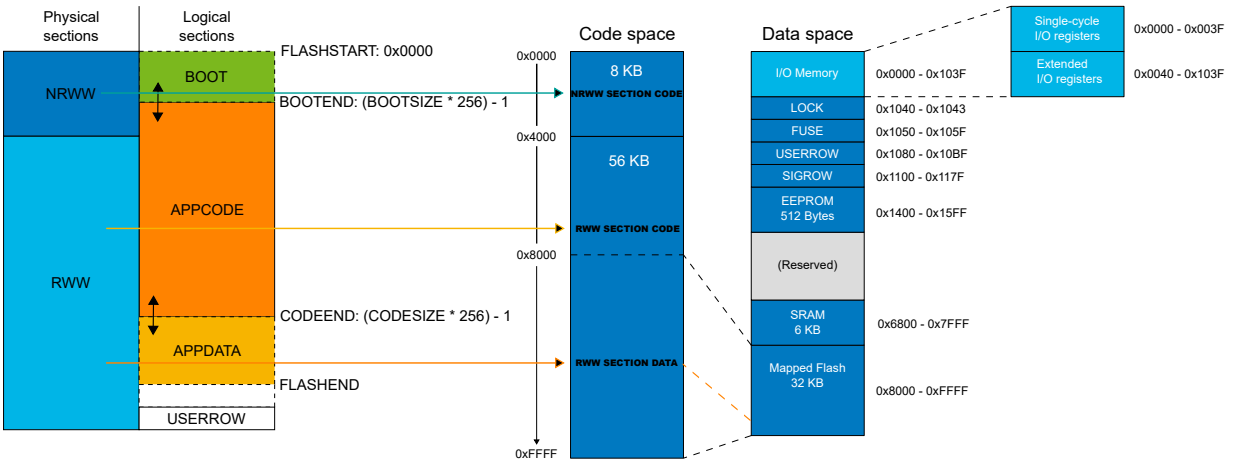
- 一个典型例子是 Bootloader 场景，CPU 可在通过通信外设（如 I²C/USART）接收主机命令的同时，对程序存储器中的 APPCODE/APPDATA 区进行编程。
- 另一个应用场景是数据记录，当模拟外设在数据可用时中断 CPU 操作，需立即将数据保存到程序存储器。

这两种场景都强调了在非易失性存储控制器（NVMCTRL）外设对程序存储器进行擦除/写入时，CPU 不应被阻塞的需求。只要确保需要在擦除/写入周期内运行的代码位于 NRWW 区，就能保证命令或中断能够及时处理，而不是等到闪存擦除/写入操作完成后再处理。

要理解如何在 AVR EA 上实现可读写同时进行（RWW）功能，可以从物理、逻辑和代码/数据空间三个角度来看程序存储器。在物理层面，程序存储器被设计为固定的 NRWW 和 RWW 区，这些分区不可更改，具体取决于器件型号——详见数据手册中的*存储器概述*章节。要实现 RWW 应用场景，需将代码和数据分别存储在对应的物理分区中。

逻辑分区包括 BOOT、APPCODE 和 APPDATA。这些分区可以通过 BOOTSIZE 和 CODESIZE 熔丝进行调整，如下图所示。注意：根据熔丝配置，物理 NRWW 区可以同时用于 BOOT 和 APPCODE/APPDATA。详细说明请参见数据手册中 NVMCTRL 外设的*存储器构成*章节。此外，若器件的程序存储器超过 32 KB，还可通过闪存映射功能在数据空间访问代码空间。详细说明请参见数据手册中 NVMCTRL 外设的*存储器外设*章节。

图 2-2. AVR64EA48闪存分区



为实现这种分区分配，需要在链接器选项中指定带有分配地址的链接器命名属性，并在声明函数和数据时使用这些命名属性。

RWW_DATA_SECTION 属性用于存放通过页写入的数据，这些数据可以从 *BOOT* 区或 *APPCODE* 区进行写入。下面的代码片段展示了如何将一个数组放置在 APPDATA 区。在编译前，请在链接器设置中定义 .rww_data 的地址。

```
// .rww_data 区域位于地址 0x2000（字地址为 0x1000）
#define RWW_DATA_SECTION __attribute__((used, section(".rww_data")))

const RWW_DATA_SECTION uint8_t rww_array[DATA_SIZE] = {0};
```

对于**可在**闪存编程或擦除过程中执行的例程（如中断、应用代码的闪存页写入或擦除，以及 Bootloader 主机通信驱动），请使用NRWW_PROG_SECTION属性。在下面的代码片段中，FillBuffer 函数被放置在NRWW区，这样在写入闪存时运行的中断就可以调用该函数。同样，需要在链接器设置中定义 .nrww_program 的地址。

```
// .nrww_program 区域位于地址 0x0400（字地址为 0x0200）
#define NRWW_PROG_SECTION __attribute__((section(".nrww_program")))

void NRWW_SECTION_CODE FillBuffer(void);
```

对于在闪存编程或擦除期间不会执行的例程，请使用 RWW_PROG_SECTION 属性。这类例程是阻塞的，必须放置在物理 RWW 区域，例如一些初始化设置函数。

下面是属性段使用的示例说明：

如果项目中使用了Bootloader，Bootloader代码应首先放置在闪存起始地址（0x0000）到BOOTEND（BOOTSIZE*256 - 1）之间。BOOT 区的大小由 BOOTSIZE 熔丝设置决定。应用代码（App code）则放在 BOOT 区之后（前提是 CODESIZE > BOOTSIZE），这意味着应用代码可能同时位于NRWW和RWW区域，从而可以将部分函数迁移到NRWW区。放置在 NRWW 区的函数或中断可以在擦除/写入RWW区时继续运行。但需要注意，必须确保在对 RWW 区进行操作期间，不能有中断或跳转到 RWW 区的代码，否则可能导致软件进入未知状态。

表 2-1. 可读写同时进行应用场景

闪存区域擦除/写入	被访问的闪存区域	CPU
NRWW区	NRWW区	暂停
RWW区	NRWW区	运行
NRWW区	RWW区	暂停
RWW区	RWW区	暂停

请在 AVR EA 的数据手册中查找闪存容量以及 NRWW/RWW 区域的相关信息。

3.在写入闪存时响应外设中断

本示例展示了从 NRWW 闪存区向 RWW 闪存区写入的优势。通过将器件引脚用于中断事件和不同的存储状态，结合示波器或逻辑分析仪，可以轻松了解代码流程及其工作原理。

测试

程序存储器被划分为更小的单元，称为页。每一页可以单独写入和擦除，由若干字节组成。对于 AVR EA，闪存以页为粒度进行操作，这意味着每当需要更改时，会对该页内的每个字节执行擦除和/或写入操作。相比之下，EEPROM 以字节为粒度，可以单独擦除和写入每个字节。

代码的工作流程是：首先擦除将要保存数据的页，然后填充缓冲区，最后使用缓冲区的数值进行页写入。一个周期性定时器中断会不断向缓冲区填充数据。

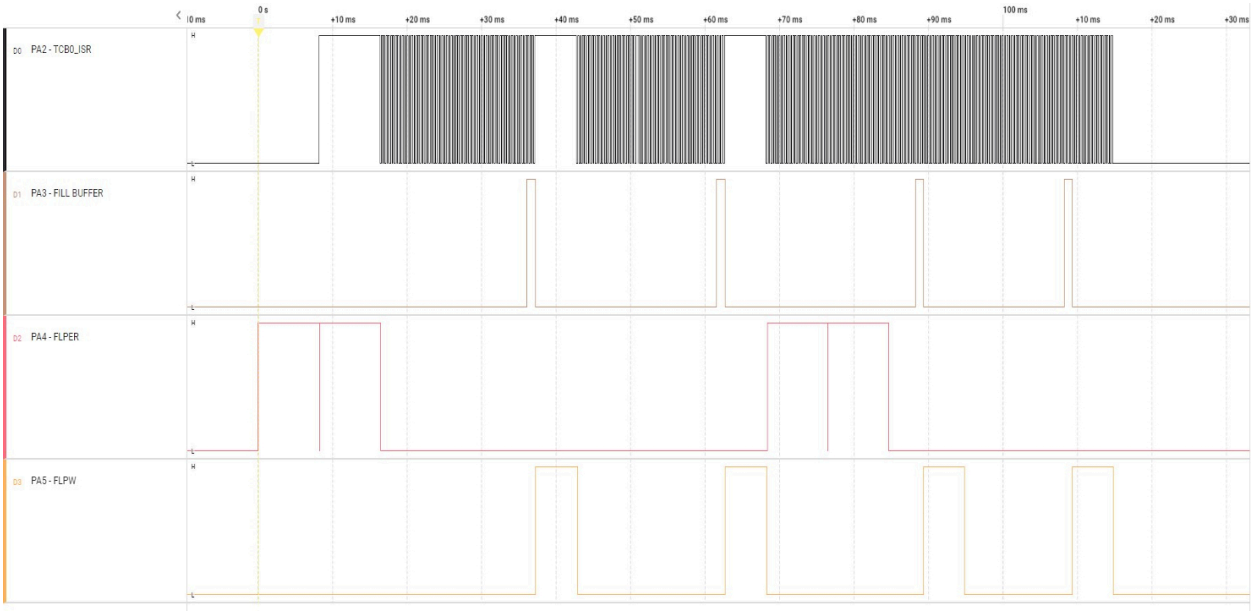
当环形缓冲区中填充的新数据量大于等于`PROGMEM_PAGE_SIZE`时，会将`PROGMEM_PAGE_SIZE`字节的数据从环形缓冲区复制到闪存页缓冲区，并启动一次闪存页写入。该过程会重复两次。

通过向 TCB 定时器/计数器的捕获/比较（CCMP）寄存器写入数据，可以改变中断服务程序（ISR）的周期。代码还会切换一些状态引脚，使得可以通过示波器等工具观察程序流程。

- 每次处理 TCB0 周期性中断 ISR 时，`PA2_ISR`引脚会切换状态
- 当闪存页缓冲区正在加载数据时，`PA3_Buffer`引脚为高电平
- 闪存页擦除进行时，`PA4_FLPER`引脚为高电平
- 闪存页写入进行时，`PA5_FLPW`引脚为高电平

随着应用程序在状态机中运行，这些引脚会不断切换状态，如下图所示。关于该流程的更详细说明，请参见示例中的“README”文件。

图 3-1. 引脚切换捕捉



理解代码

所附代码示例有两种形式——一种是在 MPLAB X Studio 中使用 Microchip代码配置器（MCC）生成驱动，另一种是在 Microchip Studio 中创建的裸机项目。两个项目的代码流程相同，分为以下几个部分：

- 主文件及主函数
- Timer 0 ——用于填充缓冲区
- Timer 1 ——用于按键消抖
- NVM 控制器

除了上述部分外，如需定义 BOOT 和 APPDATA 区域的大小，请按照下方所示设置熔丝位。

```
FUSES =
{
    .SYSCFG0 = FUSE_SYSCFG0_DEFAULT,
    .SYSCFG1 = FUSE_SYSCFG1_DEFAULT,

    .WDTCFG = FUSE_WDTCFG_DEFAULT,
    .BODCFG = FUSE_BODCFG_DEFAULT,
    .OSCCFG = FUSE_OSCCFG_DEFAULT,

    .BOOTSIZE = 31,     // 0x0000 -> 0x1F00 : BOOT Section
    .CODESIZE = 1,      // 0x01F00 -> FLASH_END : APPDATA Section
};
```

除 BOOTSIZE 和 CODESIZE 外，大多数熔丝位均保持默认值。BOOTSIZE 设置为 31 个 256 字节的块，CODESIZE 设置为 1，因此程序存储器在逻辑上被划分为 BOOT 区和 APPDATA 区。

主文件及初始化

系统的主文件包含了示例的初始化和主结构。需要注意的重要内容包括：RWW/NRWW 区域的定义方式，以及中断向量表被移动到 BOOT 区的起始位置。最后，映射的程序存储器指向第一个分区（即 .nrww_program、.nrww_data 和 .rww_data 区域的位置）。这些初始化操作确保运行的代码仅存在于程序存储器的 NRWW 区，同时在 RWW 和 NRWW 区都分配了用于数据的存储地址。

Timer 0

```
// TCB捕捉中断
TCB0.INTCTRL = TCB_CAPT_bm;
TCB0.CCMP = TOP_VALUE;
TCB0.CTRLA = TCB_CLKSEL_DIV1_gc;
```

该定时器被设置为周期性中断，在激活时会在捕捉事件发生时触发。需要注意的是，初始化过程中并未使能定时器。每当中断例程被调用时，会向数据缓冲区填充数据，具体如下所示。

```
// TCB0捕捉中断
FillBuffer();
// 清除intflag
TCB0.INTFLAGS =
TCB_CAPT_bm;
```

Timer 1

```
// TCB捕捉中断
TCB1.INTCTRL = TCB_CAPT_bm;
TCB1.CCMP = TCB1_TOP_VALUE;
TCB1.CTRLA = TCB_CLKSEL_DIV1_gc | TCB_ENABLE_bm;
```

第二个定时器同样被设置为周期性中断，并且始终处于使能状态。该中断会运行消抖函数，以避免产生误触发。

```
//   TCB1捕捉中断
DebounceSW0();
```

```
//清除intflag
TCB1.INTFLAGS = TCB_CAPT_bm;
```

擦除程序存储器

```
NRWW_DATA_last_addr = ((((uint16_t) &nrww_array) & 0x7FFF) + MAPPED_PROGMEM_START +
DATA_SIZE);

while ((uint16_t)nrwwFlashPointer < NRWW_DATA_last_addr)
{
    _PROTECTED_WRITE_SPM(NVMCTRL.CTRLA, NVMCTRL_CMD_NOCMD_gc);

        // 对该地址进行一次虚写，以更新 NVMCTL 中的地址寄存器
    *nrwwFlashPointer = 0;

    _PROTECTED_WRITE_SPM(NVMCTRL.CTRLA, NVMCTRL_CMD_FLPER_gc);

    while (NVMCTRL.STATUS & NVMCTRL_FLBUSY_bm)
    {
        ; // 等待闪存擦除完成
    }
    SCOPE_PORT.OUTCLR = SCOPE_FLPER_bm;

    nrwwFlashPointer = nrwwFlashPointer + PROGMEM_PAGE_SIZE;
}
```

NRWW 和 RWW 数据区的闪存擦除操作是相同的。操作开始时，首先将闪存页缓冲区指向待擦除的区域。此时，FLPER 引脚被拉高，因此可以通过示波器观察到闪存擦除操作即将开始。随后，向 NVM 控制器发送 NOCMD 指令，以清除之前的 NVM 命令，避免命令冲突错误，并进行一次虚写/存储操作。接着，将页擦除命令发送给 NVM 控制器。当 NVMCTRL.STATUS 寄存器显示闪存不再忙碌时，FLPER 引脚被拉低，并将下一个擦除地址按页大小递增。数据区写入完成后，闪存擦除操作即告结束。

写入程序存储器

```
// 检查是否有一整页需要填充
if ((writeIndex - readIndex) >= PROGMEM_PAGE_SIZE)
{
    _PROTECTED_WRITE_SPM(NVMCTRL.CTRLA, NVMCTRL_CMD_NOCMD_gc);

    for (uint8_t i = 0; i < PROGMEM_PAGE_SIZE; i++)
    {
        *nrwwFlashPointer++ = buffer[readIndex++];
     }
     SCOPE_PORT.OUTCLR = SCOPE_BUFFER_bm;
     SCOPE_PORT.OUTSET = SCOPE_FLPW_bm;

    // 写入闪存页
    _PROTECTED_WRITE_SPM(NVMCTRL.CTRLA, NVMCTRL_CMD_FLPW_gc);
    while (NVMCTRL.STATUS & NVMCTRL_FLBUSY_bm)
    {
        ; // 等待闪存页写入操作完成
    }
}
```

NRWW 和 RWW 数据区的闪存编程操作是相同的。操作开始时，首先检查是否有足够的数据可以写入一整页闪存。如果数据不足，则会等待直到生成足够的数据。如果数据充足，则向 NVM 控制器发送 NOCMD 指令，以清除之前的 NVM 命令，避免命令冲突错误。随后，BUFFER 引脚被拉高，因此可以通过示波器观察到缓冲区加载操作即将开始。数据会从数据缓冲区读取并存储到闪存页缓冲区。当一整页的数据都已
读取并存储后，BUFFER 引脚被拉低。接着，FLPW 引脚被拉高，并将页写入命令发送给 NVM 控制器。当 NVMCTRL.STATUS 寄存器显示闪存不再忙碌时，FLPW 引脚被拉低，并将下一个页写入地址按页大小递增。数据区写入完成后，闪存写入操作即告结束。

上述擦除和写入功能的代码片段来自裸机示例，在 MCC 示例中格式会有所不同，但每个示例的功能是相同的。

填充缓冲区

```
void
FillBuffer(void){
    // 向缓冲区填充一些数据
    SCOPE_PORT.OUTTGL = SCOPE_ISR_bm;
    buffer[writeIndex++] = (data >> 2);
    data++;
    // 检查是否发生缓冲区溢出
    if (writeIndex == readIndex)
{
        // 溢出
        SCOPE_PORT.OUTTGL = SCOPE_OVERFLOW_bm;
    }
}
```

当 Timer 0 的比较计数器达到设定的 TOP 值时，周期性中断会触发 FillBuffer 函数。此时，ISR 引脚会切换状态，缓冲区被填充数据。如果缓冲区发生溢出，OVERFLOW 引脚也会切换状态。

SW0 操作与消抖

NRWW 和 RWW 数据区的擦除与写入操作均由按下 SW0 按键启动。之后每次按下 SW0 都会触发相同的流程。RWW 数据区最后一个数据位置的数值会作为新数据的种子。由于这种机制，当按下 SW0 并读取程序存储器时，用户可以观察到 NRWW 和 RWW 数据区的数据发生变化。作为 SW0 被按下的确认，LED0 会点亮。需要通过按键触发整个示例操作，以便用户能够提前设置好外部仪器（如示波器和/或逻辑分析仪），用于捕捉各监控引脚的切换信号。为此，采用并实现了一个简单的消抖算法，TCB1 用于该功能。对物理按键 SW0 进行软件消抖，可以防止擦除和写入操作被误触发。请注意，编程闪存的擦除/写入次数是有限的，超过规定次数后操作无法保证（详细信息请参阅器件的电气特性说明）。

中断向量表迁移

中断向量的放置位置取决于 Control A（CPUINT.CTRLA）寄存器中的中断向量选择（IVSEL）位的取值。如果没有使用中断源，常规程序代码可以放在这些位置。当 IVSEL 位置 1 时，中断向量会被放置在闪存 BOOT 区的起始位置，这样在 RWW 区进行擦除/写入操作时，中断仍能正常运行。如前文概述所述，“RWW 区”这一术语指的是正在被编程（擦除或写入）的区域，而不是正在被读取的区域。只有位于 NRWW 闪存区的代码，才能在对 RWW 区进行编程时通过执行 CPU 指令或读取数据来访问。

4. 从MPLAB Discover获取代码示例

MPLAB Discover 将来自多个渠道的可用资源整合到一个门户中。

MPLAB Discover网页：MPLAB Discover

代码示例

- Microchip Studio代码示例
- MPLAB X代码示例

您可以直接从 MPLAB Discover 下载代码示例的`.zip`文件。以“*studio*”结尾的代码库可在 Microchip Studio 中打开，以 “*mplab-mcc*”结尾的代码库可在 MPLAB X 中打开。

当代码示例托管在 GitHub 上时，MPLAB Discover 会提供“*Open with Github*”链接。您可以通过该链接下载示例代码，或使用电脑上的 `git` 工具创建本地代码库克隆。

5. 版本历史

版本历史	日期	备注
A	10/2023	本文档的初始版本

Microchip信息

商标

“Microchip”的名称和徽标组合、“M”徽标及其他名称、徽标和品牌均为 Microchip Technology Incorporated 或其关联公司和/或子公司在美国和/或其他国家或地区的注册商标或商标（“Microchip 商标”）。有关 Microchip 商标的信息，可访问 https://www.microchip.com/en-us/about/legal-information/microchip-trademarks。

ISBN：979-8-3371-1009-7

法律声明

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

本出版物及其提供的信息仅适用于 Microchip 产品，包括设计、测试以及将 Microchip 产品集成到您的应用中。以其他任何方式使用这些信息都将被视为违反条款。本出版物中的器件应用信息仅为您提供便利，将来可能会发生更新。您须自行确保应用符合您的规范。如需额外的支持，请联系当地的 Microchip 销售办事处，或访问 www.microchip.com/en-us/support/design-help/client-support-services。

Microchip“按原样”提供这些信息。Microchip 对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对非侵权性、适销性和特定用途的适用性的暗示担保，或针对其使用情况、质量或性能的担保。

在任何情况下，对于因这些信息或使用这些信息而产生的任何间接的、特殊的、惩罚性的、偶然的或附带的损失、损害或任何类型的开销，Microchip 概不承担任何责任，即使 Microchip 已被告知可能发生损害或损害可以预见。在法律允许的最大范围内，对于因这些信息或使用这些信息而产生的所有索赔，Microchip 在任何情况下所承担的全部责任均不超出您为获得这些信息向 Microchip 直接支付的金额（如有）。如果将 Microchip 器件用于生命维持和/或生命安全应用，一切风险由买方自负。买方同意在由此引发任何一切损害、索赔、诉讼或费用时，会维护和保障 Microchip 免于承担法律责任。除非另外声明，在 Microchip 知识产权保护下，不得暗中或以其他方式转让任何许可证。

Microchip 器件代码保护功能

请注意以下有关 Microchip 产品代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术规范。
- Microchip 确信：在按照操作规范正常使用的情况下，Microchip 系列产品非常安全。
- Microchip 重视并积极保护其知识产权。任何试图破坏 Microchip 产品代码保护功能的行为均可视为违反了《数字器件千年版权法案（Digital Millennium Copyright Act）》并予以严禁。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。