MICROCHIP

TB3186

如何使用 Cortex®-M 高速缓存控制器实现确定性代码性能

简介

在基于单片机的嵌入式应用中，软件存储在非易失性存储器（通常是闪存）中并从中运行。闪存虽然为存储和执行代码提供了一种有效的介质，但从闪存中执行时，许多因素会限制确定性代码性能。影响确定性代码行为的一个重要因素是系统总线矩阵的复杂性。从 SRAM 中运行代码时，由于与闪存相同的原因，也会出现确定性代码性能问题。

非确定性代码性能主要是由于代码从存储器传播到 CPU 的时间不同造成的，而传播时间的不同与连接存储器和 CPU 的系统总线矩阵有关。如果系统中存在多个实体需要访问系统总线，非确定性代码性能问题将会更加明显。

在实时应用中，某些情况下会有一些小的、关键的代码片段要求限时执行。不建议从闪存或 SRAM 中运行这类代码，因为系统总线仲裁可能会导致达不到预期的确定性时序，也就是说，从闪存或 SRAM 中提取代码时出现了高速缓存未命中情况。代码访问时间会因系统总线访问的可用性而异，因为需要在多个总线实体之间进行仲裁。

实现关键代码确定性代码性能的有效方法是将代码从紧耦合存储器（Tightly Coupled Memory，TCM）传播到处理器执行，以避免高速缓存未命中的情况。基于 Microchip Cortex®-M4 的单片机（MCU）上的 Arm® Cortex-M 高速缓存控制器（CMCC）支持从高速缓存存储器中运行关键代码以实现确定性性能。

目录

简介..... 1

1. 概念..... 3

2. 解决方案..... 4

3. 确定性性能分析..... 10

4. 相关资源..... 12

Microchip 网站..... 13

变更通知客户服务..... 13

客户支持..... 13

Microchip 器件代码保护功能..... 13

法律声明..... 14

商标..... 14

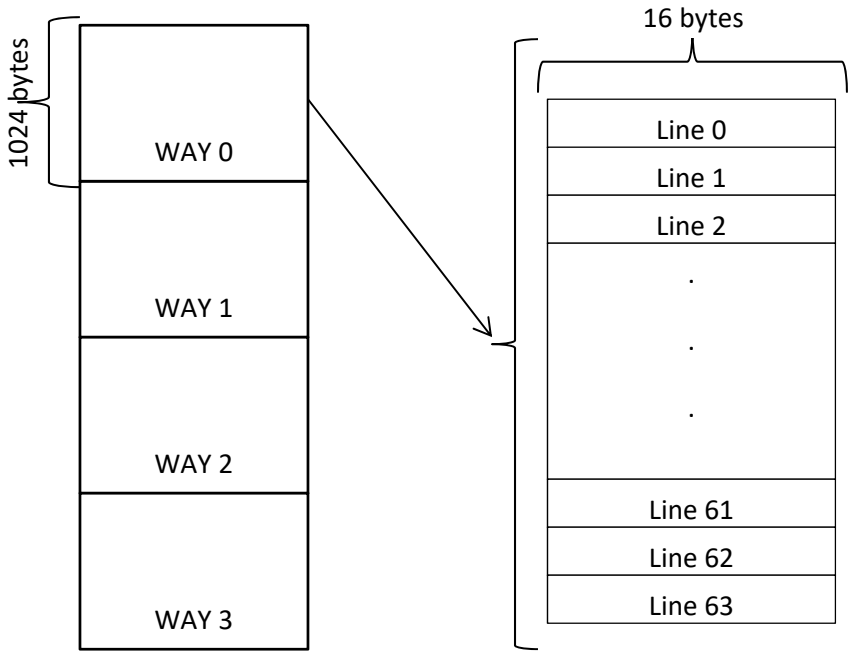
DNV 认证的质量管理体系..... 15

全球销售及服务网点..... 16

1. 概念

基于 Cortex-M4 的 MCU（即 SAME54）上的 CMCC 具有一个 4 KB 的专用四路 L1 组相联高速缓存，如下图所示。

图 1-1. 四路 L1 组相联高速缓存存储器

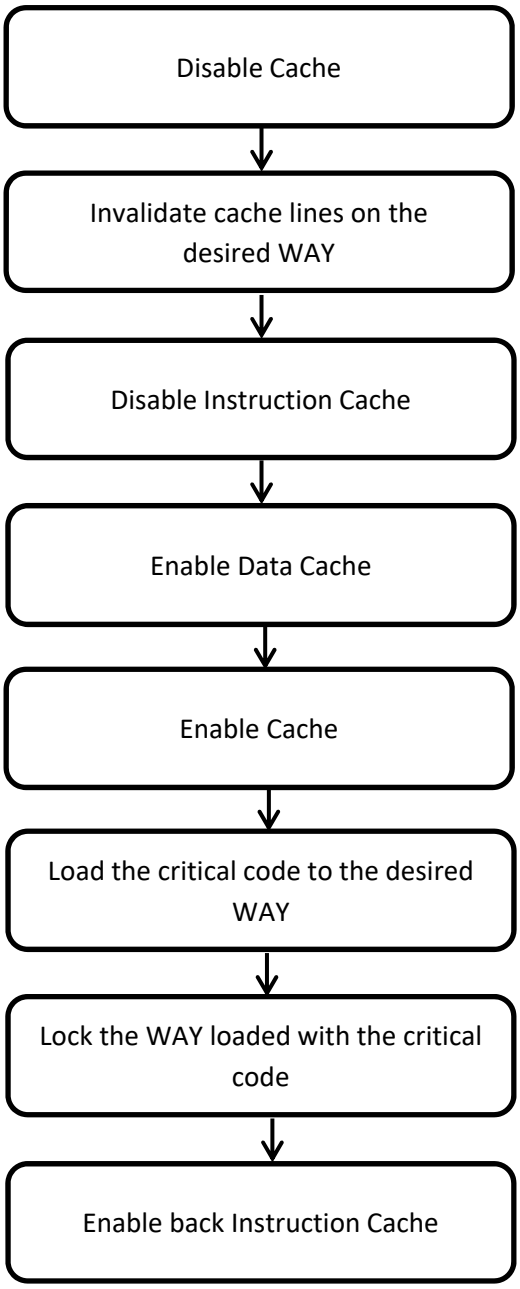


借助 CMCC，可将关键代码加载到一个高速缓存*路*中并锁定，从而将此部分高速缓存用作 TCM 以实现确定性代码性能。特定的*路*被锁定时，CMCC 将不会使用该锁定的*路*来处理常规缓存事务。加载了关键代码并被锁定的该高速缓存*路*能始终实现高速缓存命中。

2. 解决方案

以下流程序列展示了用于实现确定性代码性能的代码。

图 2-1. 实现确定性代码性能的流程序列



以下代码示例展示了如何实现提供确定性代码性能的函数及如何在基于 Cortex-M4 的 MCU（即 SAME54）上使用该函数。

图 2-2. cmcc_loadnlock 函数中使用的宏和变量的实现

```
#define   CMCC_NO_OF_WAYS            4
#define   CMCC_WAYSIZE               1024
#define   CMCC_WAY_NO_OF_LINES       64
#define   CMCC_WAY_LINE_SIZE         16

#define   CMCC_WAY0                 (1 << CMCC_MAINT1_WAY_WAY0_Val)
#define   CMCC_WAY1                 (1 << CMCC_MAINT1_WAY_WAY1_Val)
#define   CMCC_WAY2                 (1 << CMCC_MAINT1_WAY_WAY2_Val)
#define   CMCC_WAY3                 (1 << CMCC_MAINT1_WAY_WAY3_Val)

const uint8_t load_pass[CMCC_WAY_LINE_SIZE] = { 0xA5, 0x5A, 0xA5, 0x5A,
                                                0xA5, 0x5A, 0xA5, 0x5A,
                                                0xA5, 0x5A, 0xA5, 0x5A,
                                                0xA5, 0x5A, 0xA5, 0x5A };
```

图 2-3. cmcc_loadnlock 函数实现

```
void cmcc_loadnlock(uint32_t way_bitfield, void (*f)(void), uint32_t size)
{
    volatile uint32_t dummy;
    uint8_t* p_foo;
    int8_t way_index;

    /* Disable the cache */
    CMCC->CTRL.reg = 0;
    while (CMCC->SR.bit.CSTS != 0);
    /* Invalidate by line the whole desired WAY */
    for (volatile uint32_t i = 0; i < CMCC_WAY_NO_OF_LINES; i++) {
        CMCC->MAINT1.reg = CMCC_MAINT1_WAY(way_bitfield) | CMCC_MAINT1_INDEX(i);
    }

    /* Disable instruction cache (icdis register is set ) */
    CMCC->CFG.bit.ICDIS = 1;

    /* Enable data cache (dcdis register is clear) */
    CMCC->CFG.bit.DCDIS = 0;

    /* Enable the cache */
    CMCC->CTRL.reg = CMCC_CTRL_CEN;
    while (CMCC->SR.reg != CMCC_SR_CSTS);
    /* Find the WAY index */
    if(CMCC_WAY3 == way_bitfield) {
        way_index = ((way_bitfield >> 1 ) - 1);
    } else {
        way_index = way_bitfield >> 1;
    }

    /* Parse through the WAYs to find the desired WAY */
    for(uint32_t indx = 0; indx < CMCC_NO_OF_WAYS; indx++) {
        if(way_index != indx) {
            /* Not the desired WAY, Still need to pass through by
            loading the entire WAY by loading dummy data */
            for (uint32_t i = 0; i < CMCC_WAY_NO_OF_LINES; i++) {
                dummy = *(uint8_t *) load_pass;
            }
        } else {
            /* Desired WAY found, Load with the critical code from flash */
            size /= CMCC_WAY_LINE_SIZE;
            p_foo = (uint8_t *)f;
            for (uint32_t i = 0; i < size; i++) {
                dummy = *p_foo;
                p_foo += CMCC_WAY_LINE_SIZE;
            }
            break;
        }
    }

    /* Then write the lock per way register */
    CMCC->LCKWAY.bit.LCKWAY |= way_bitfield;

    /* Re-enable instruction cache (icdis register is clear ) */
    CMCC->CFG.bit.ICDIS = 0;
}
```

上述函数实现中的变量 CMCC 定义如下例所示。此处提供的实现适用于基于 Cortex-M4 的器件（即 SAME54P20A）。Microchip Atmel START（ASF4）库中提供了关于结构实现的详细信息以及 CMCC_MAINT1_WAY、CMCC_MAINT1_INDEX、CMCC_CTRL_CEN 和 CMCC_SR_CSTS 宏的定义。

图 2-4. CMCC 结构声明

```
#define CMCC              ((Cmcc     *)0x41006000UL) /**< \brief (CMCC) APB Base Address */
typedef struct {
  __I  CMCC_TYPE_Type           TYPE;         /**< \brief Offset: 0x00 (R/  32) Cache Type Register */
  __IO CMCC_CFG_Type            CFG;          /**< \brief Offset: 0x04 (R/W 32) Cache Configuration Register */
  __O  CMCC_CTRL_Type           CTRL;         /**< \brief Offset: 0x08 ( /W 32) Cache Control Register */
  __I  CMCC_SR_Type             SR;           /**< \brief Offset: 0x0C (R/  32) Cache Status Register */
  __IO CMCC_LCKWAY_Type         LCKWAY;       /**< \brief Offset: 0x10 (R/W 32) Cache Lock per Way Register */
       RoReg8                   Reserved1[0xC];
  __O  CMCC_MAINT0_Type         MAINT0;       /**< \brief Offset: 0x20 ( /W 32) Cache Maintenance Register 0 */
  __O  CMCC_MAINT1_Type         MAINT1;       /**< \brief Offset: 0x24 ( /W 32) Cache Maintenance Register 1 */
  __IO CMCC_MCFG_Type           MCFG;         /**< \brief Offset: 0x28 (R/W 32) Cache Monitor Configuration Register */
  __IO CMCC_MEN_Type            MEN;          /**< \brief Offset: 0x2C (R/W 32) Cache Monitor Enable Register */
  __O  CMCC_MCTRL_Type          MCTRL;        /**< \brief Offset: 0x30 ( /W 32) Cache Monitor Control Register */
  __I  CMCC_MSR_Type            MSR;          /**< \brief Offset: 0x34 (R/  32) Cache Monitor Status Register */
} Cmcc;
```

实现

请参考以下步骤实现 cmcc_loadnlock 函数：

1. 使能高速缓存后，CMCC 以轮询方式使用四*路*组相联高速缓存，首先从 *WAY0* 开始。这在使能高速缓存后立即开始。
2. `cmcc_loadnlock` 函数扫描各高速缓存*路*以找到所需的*路*（参见标记为 A 的代码示例）。
3. 在单次运行中，如果未找到所需的高速缓存*路*，`cmcc_loadnlock` 函数将数组中的值加载到整个高速缓存*路*，以越过正在处理的*路*并继续检查下一*路*是否为所需的高速缓存*路*（参见标记为 B 的代码示例）。
4. 找到所需的高速缓存*路*后，`cmcc_loadnlock` 函数将从闪存读取关键代码到虚拟变量，以确保关键代码存储在该高速缓存*路*中（参见标记为 C 的代码示例）。
5. 在标记为 B 和 C 的代码中，`cmcc_loadnlock` 函数只加载所需的 4 个字（16 字节）中的一个字（4 字节），这是因为 AHB 的 WRAP4 特性通过加载包含 16 字节的整行来填补空白。
6. `cmcc_loadnlock` 函数只锁定所需的高速缓存*路*，以捕获高速缓存中的关键代码。在锁定特定的高速缓存*路*后，CMCC 将不再使用该锁定的*路*来处理常规的轮询高速缓存事务。锁定一个高速缓存*路*并在高速缓存中捕获关键代码的过程模拟了调用关键代码时获得 100%高速缓存命中的情况。
7. 上述显示的 `cmcc_loadnlock` 函数实现适用于关键代码大小不超过高速缓存*路*大小（`CMCC_WAYSIZE`，即 1024 字节）的情况。如果关键代码大小超过了高速缓存*路*的大小，则需要使用连续的第二*路*来存放关键代码。若想容纳更大的关键代码，则需要增强 `cmcc_loadnlock` 函数实现。

   在增强 `cmcc_loadnlock` 函数实现时，请注意以下准则。

   7.1. 对参数 ***size*** 进行评估，以确定关键代码是否能够存放在 4 KB 的可用高速缓存中。如果能够存放在可用高速缓存中，函数实现应确定需要多少高速缓存*路*。

   7.2. 函数实现需确定是否能为关键代码分配所需数目的连续高速缓存*路*。

   7.3. 此实现会使连续的多个高速缓存*路*失效（参见标记为 1 的代码示例）。

   7.4. 此实现会加载包含关键代码的多个连续高速缓存*路*（参见标记为 C 的代码示例）。

   7.5. 此实现会锁定连续的多个高速缓存*路*以捕获高速缓存中的代码（参见标记为 2 的代码示例）。

用法

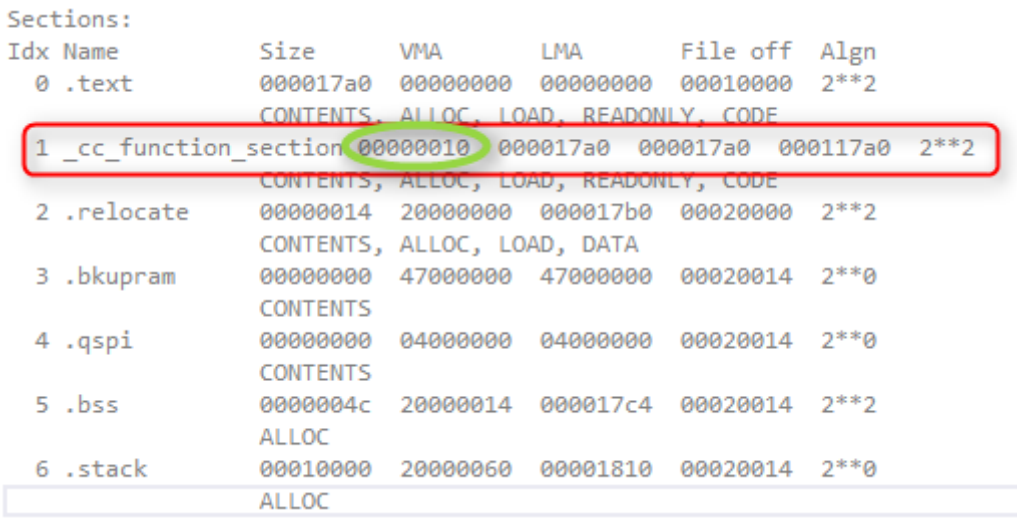
以下代码示例展示了 cmcc_loadnlock 函数的用法。

图 2-5. cmcc_loadnlock 函数的用法

```
static void _critical_code_function(void) __attribute__ ((section ("_cc_function_section")));
int main(void)
{
    atmel_start_init();

    cmcc_enable();
    cmcc_loadnlock(CMCC_WAY0, _critical_code_function, 0x10);

    _critical_code_function();

    while (true)
    {
        ;
    }
}

static void _critical_code_function(void)
{
    /* Implement your critical code */
}
```

1. cmcc_loadnlock 函数接受三个参数。
    1.1. 第一个参数 way_bitfield 是高速缓存*路*的数量，它将从四个*路*（CMCC_WAY0、CMCC_WAY1、CMCC_WAY2、CMCC_WAY3）的值中取一个值。
    1.2. 第二个参数 f 是始终需要从高速缓存运行的关键代码函数的地址。
    1.3. 第三个参数 size 是关键代码函数的大小。该大小不应超过高速缓存*路*的大小（*CMCC_WAYSIZE*）。如果超过了高速缓存*路*的大小，则按照上述步骤 7 中所述增强实现。
2. 关键代码函数（_critical_code_function）的确切大小可从项目的清单文件中获得。使用以下步骤查找关键代码函数的大小。
    2.1. 在前面的示例中，关键代码函数使用段属性_cc_function_section 进行声明。这是对 GNU 链接器的一条指令，用于将函数_critical_code_function 放入名为 _cc_function_section 的代码段中。
    2.2. 在 cmcc_loadnlock 函数调用中设置虚拟数据大小。
    2.3. 清理并构建项目。
    2.4. 打开项目的清单文件（.lss）并找到链接器标记_cc_function_section 。该标记所在的行显示了段大小。这就是关键代码函数的大小。

图 2-6. 找到_cc_function_section 属性



2.5. 使用`.lss`文件中的大小替换调用函数`cmcc_loadnlock`中的关键代码函数大小。

2.6. 编译和编程

3. `cmcc_loadnlock`函数可以被连续调用多次，从而在多个高速缓存*路*中动态存储多个关键代码函数。

注：

1. 本文档仅介绍如何使用统一的四*路*L1 组相联高速缓存来实现确定性代码性能优化。CMCC 还允许数据缓存并将部分高速缓存用作数据 TCM。有关其他信息，请参见基于 Cortex-M4 的 MCU（SAME54）数据手册。
2. 本文档中讨论的确定性代码性能实现以减少活动高速缓存大小为代价。

3. 确定性性能分析

在使能了高速缓存的系统中，代码性能取决于高速缓存命中或未命中的频率。高速缓存命中率越高，性能就越好。例如，在一个简单的应用中，只有一个函数以迭代方式执行关键运算，这种情况下高速缓存的命中率就较高。相反，在复杂的实时应用中，会有多个实体（例如系统总线主器件）访问非易失性存储器，由于非确定性调用序列可能会导致高速缓存未命中率较高，因此代码性能会受限。此外，系统总线矩阵还会导致一定的延迟。

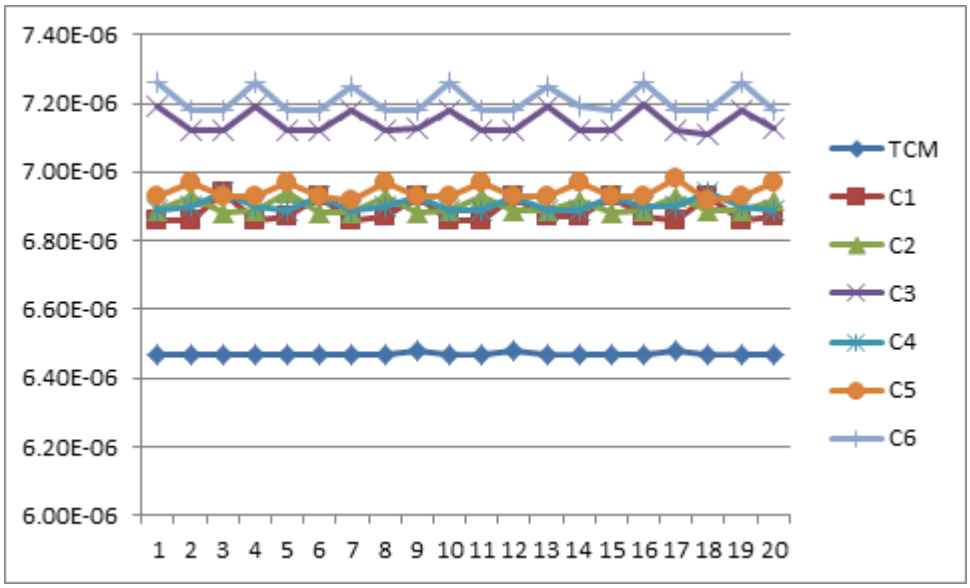
下面将对一个简单应用程序分别通过 TCM（使用前面讨论的 `loadnlock` 实现）和常规高速缓存事务运行代码时的性能进行对比分析。

- 以下 1 KB 大小的函数均包含相同的代码以用于性能分析。

```
critical_code_function (); // [Referred as TCM]
_function_1 (); // [Referred as C1]
_function_2 (); // [Referred as C2]
_function_3 (); // [Referred as C3]
_function_4 (); // [Referred as C4]
_function_5 (); // [Referred as C5]
_function_6 (); // [Referred as C6]
```

- 关键代码函数[TCM]始终从高速缓存中运行（通过调用 `loadnlock` 被锁定在某高速缓存*路*中），其他函数从闪存中运行并通过 CMCC 基于轮询调度序列放入高速缓存中。
- 上述函数都按相同顺序被无限调用。
- 下图显示了在 120 MHz、15 NVM 等待状态且无代码优化的情况下，在基于 Cortex-M4 的处理器（SAME54MCU）上运行的上述函数序列的 20 个样本分析结果。图中纵轴显示的时间以微秒（µs）为单位。

图 3-1. 确定性代码性能分析



- 总结：
  - 从 TCM 运行代码所花费的时间是确定的，需要 6.47 或 6.48 µs。这与从常规高速缓存事务运行代码所花费的非确定性时间形成了鲜明对比。对于单次运行，例如 Pass1，从常规高速缓存事务运行代码所消耗的时间从 6.86 µs 到 7.26 µs 不等。
  - 运行 C3 和 C6 函数时可能会出现高速缓存未命中情况。在运行这些函数时，消耗的时间出现了突然增加（C3 耗时 7.19 µs，C6 耗时 7.26 µs）。

- 从 TCM 运行代码与从高速缓存事务运行代码相比，除了具有确定性代码性能之外，还具有性能优势。对于简单应用程序，在单次运行（例如 Pass1）中，从 TCM 运行代码所花费的时间为 6.47μs，而通过正常高速缓存事务运行代码所花费的时间则从 6.86 μs 到 7.26 μs 不等。

上述讨论的代码 TCM 实现为关键代码提供了确定性的性能模型。与使能了高速缓存的系统相比，它还具有性能优势。性能优势与应用程序的复杂性有关。随着应用程序复杂性的增加，从 TCM 执行代码的性能优势会更明显。

4. 相关资源

有关更多信息，请参见以下文档：

- SAM D5x/E5x 系列数据手册：
  http://ww1.microchip.com/downloads/en/DeviceDoc/60001507A.pdf
- SMART SAM E70 TCM 存储器：
  http://ww1.microchip.com/downloads/en/AppNotes/atmel-42555-smart-sam-e70-tcm-memory_application%20note_at14971.pdf
- 如何优化 SAM S70/E70/V7x 架构的使用：
  http://ww1.microchip.com/downloads/en/appnotes/atmel-44047-cortex-m7-microcontroller-optimize-usage-sam-v71-v70-e70-s70-architecture_application-note.pdf

Microchip 网站

Microchip 网站 http://www.microchip.com/为客户提供在线支持。客户可通过该网站方便地获取文件和信息。只要使用常用的互联网浏览器即可访问，网站提供以下信息：

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

变更通知客户服务

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

欲注册，请登录 Microchip 网站 http://www.microchip.com/。在“支持”（Support）下，点击“变更通知客户”（Customer Change Notification）服务后按照注册说明完成注册。

客户支持

Microchip 产品的用户可通过以下渠道获得帮助：

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

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

也可通过以下网站获得技术支持： http://www.microchip.com/support

Microchip 器件代码保护功能

请注意以下有关 Microchip 器件代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术指标。
- Microchip 确信：在正常使用的情况下，Microchip 系列产品是当今市场上同类产品中最安全的产品之一。
- 目前，仍存在着恶意、甚至是非法破坏代码保护功能的行为。就我们所知，所有这些行为都不是以 Microchip 数据手册中规定的操作规范来使用 Microchip 产品的。这样做的人极可能侵犯了知识产权。
- Microchip 愿意与关心代码完整性的客户合作。
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。

代码保护功能处于持续发展中。Microchip 承诺将不断改进产品的代码保护功能。任何试图破坏 Microchip 代码保护功能的行为均可视为违反了《数字器件千年版权法案（Digital Millennium Copyright Act）》。如

果这种行为导致他人在未经授权的情况下，能访问您的软件或其他受版权保护的成果，您有权依据该法案提起诉讼，从而制止这种行为。

法律声明

本出版物中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。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，包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。Microchip 对因这些信息及使用这些信息而引起的后果不承担任何责任。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伤害、索赔、诉讼或费用时，会维护和保障 Microchip 免于承担法律责任，并加以赔偿。除非另外声明，否则在 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。

商标

Microchip 的名称和徽标组合、Microchip 徽标、AnyRate、AVR、AVR 徽标、AVR Freaks、BitCloud、chipKIT、chipKIT 徽标、CryptoMemory、CryptoRF、dsPIC、FlashFlex、flexPWR、Heldo、JukeBlox、KeeLoq、Kleer、LANCheck、LINK MD、maXStylus、maXTouch、MediaLB、megaAVR、MOST、MOST 徽标、MPLAB、OptoLyzer、PIC、picoPower、PICSTART、PIC32 徽标、Prochip Designer、QTouch、SAM-BA、SpyNIC、SST、SST 徽标、SuperFlash、tinyAVR、UNI/O 和 XMEGA 是 Microchip Technology Incorporated 在美国和其他国家或地区的注册商标。

ClockWorks、The Embedded Control Solutions Company、EtherSynch、Hyper Speed Control、HyperLight Load、IntelliMOS、mTouch、Precision Edge 和 Quiet-Wire 为 Microchip Technology Incorporated 在美国的注册商标。

Adjacent Key Suppression、AKS、Analog-for-the-Digital Age、Any Capacitor、AnyIn、AnyOut、BodyCom、CodeGuard、CryptoAuthentication、CryptoAutomotive、CryptoCompanion、CryptoController、dsPICDEM、dsPICDEM.net、Dynamic Average Matching、DAM、ECAN、EtherGREEN、In-Circuit Serial Programming、ICSP、INICnet、Inter-Chip Connectivity、JitterBlocker、KleerNet、KleerNet 徽标、memBrain、Mindi、MiWi、motorBench、MPASM、MPF、MPLAB Certified 徽标、MPLIB、MPLINK、MultiTRAK、NetDetach、Omniscient Code Generation、PICDEM、PICDEM.net、PICkit、PICtail、PowerSmart、PureSilicon、QMatrix、REAL ICE、Ripple Blocker、SAM-ICE、Serial Quad I/O、SMART-I.S.、SQI、SuperSwitcher、SuperSwitcher II、Total Endurance、TSHARC、USBCheck、VariSense、ViewSpan、WiperLock、Wireless DNA 和 ZENA 为 Microchip Technology Incorporated 在美国和其他国家或地区的商标。

SQTP 为 Microchip Technology Inc.在美国的服务标记。

Silicon Storage Technology 为 Microchip Technology Inc.在除美国外的国家或地区的注册商标。

GestIC 是 Microchip Technology Inc.的子公司 Microchip Technology Germany II GmbH & Co. KG 在除美国外的国家或地区的注册商标。

在此提及的所有其他商标均为各持有公司所有。

ISBN： 978-1-5224-3500-6

AMBA、Arm、Arm7、Arm7TDMI、Arm9、Arm11、Artisan、big.LITTLE、Cordio、CoreLink、CoreSight、Cortex、DesignStart、DynamIQ、Jazelle、Keil、Mali、Mbed、Mbed Enabled、NEON、

DNV 认证的质量管理体系

ISO/TS 16949

Microchip 位于美国亚利桑那州 Chandler 和 Tempe 与位于俄勒冈州 Gresham 的全球总部、设计和晶圆生产厂及位于美国加利福尼亚州和印度的设计中心均通过了 ISO/TS-16949:2009 认证。Microchip 的 PIC® MCU 和 dsPIC® DSC、KEELOQ®跳码器件、串行 EEPROM、单片机外设、非易失性存储器及模拟产品严格遵守公司的质量体系流程。此外，Microchip 在开发系统的设计和生产方面的质量体系也已通过了 ISO 9001:2000 认证。

全球销售及服务网点

美洲	亚太地区	亚太地区	欧洲
公司总部 2355 West Chandler Blvd. Chandler, AZ 85224-6199 电话：1-480-792-7200 传真：1-480-792-7277 技术支持： http://www.microchip.com/support 网址： www.microchip.com **亚特兰大** 德卢斯，乔治亚州 电话：1-678-957-9614 传真：1-678-957-1455 **奥斯汀，德克萨斯州** 电话：1-512-257-3370 **波士顿** 韦斯特伯鲁，马萨诸塞州 电话：1-774-760-0087 传真：1-774-760-0088 **芝加哥** 艾塔斯卡，伊利诺伊州 电话：1-630-285-0071 传真：1-630-285-0075 **达拉斯** 艾迪生，德克萨斯州 电话：1-972-818-7423 传真：1-972-818-2924 **底特律** 诺维，密歇根州 电话：1-248-848-4000 **休斯敦，德克萨斯州** 电话：1-281-894-5983 **印第安纳波利斯** 诺布尔斯维尔，印第安纳州 电话：1-317-773-8323 传真：1-317-773-5453 电话：1-317-536-2380 **洛杉矶** 米申维耶霍，加利福尼亚州 电话：1-949-462-9523 传真：1-949-462-9608 电话：1-951-273-7800 **罗利，北卡罗来纳州** 电话：1-919-844-7510 **纽约，纽约州** 电话：1-631-435-6000 **圣何塞，加利福尼亚州** 电话：1-408-735-9110 电话：1-408-436-4270 **加拿大 - 多伦多** 电话：1-905-695-1980 传真：1-905-695-2078	**中国 - 北京** 电话：86-10-8569-7000 **中国 - 成都** 电话：86-28-8665-5511 **中国 - 重庆** 电话：86-23-8980-9588 **中国 - 东莞** 电话：86-769-8702-9880 **中国 - 广州** 电话：86-20-8755-8029 **中国 - 杭州** 电话：86-571-8792-8115 **中国 - 南京** 电话：86-25-8473-2460 **中国 - 青岛** 电话：86-532-8502-7355 **中国 - 上海** 电话：86-21-3326-8000 **中国 - 沈阳** 电话：86-24-2334-2829 **中国 - 深圳** 电话：86-755-8864-2200 **中国 - 苏州** 电话：86-186-6233-1526 **中国 - 武汉** 电话：86-27-5980-5300 **中国 - 西安** 电话：86-29-8833-7252 **中国 - 厦门** 电话：86-592-2388138 **中国 - 香港特别行政区** 电话：852-2943-5100 **中国 - 珠海** 电话：86-756-3210040 **台湾地区 - 高雄** 电话：886-7-213-7830 **台湾地区 - 台北** 电话：886-2-2508-8600 **台湾地区 - 新竹** 电话：886-3-577-8366	**澳大利亚 - 悉尼** 电话：61-2-9868-6733 **印度 - 班加罗尔** 电话：91-80-3090-4444 **印度 - 新德里** 电话：91-11-4160-8631 **印度 - 浦那** 电话：91-20-4121-0141 **日本 - 大阪** 电话：81-6-6152-7160 **日本 - 东京** 电话：81-3-6880- 3770 **韩国 - 大邱** 电话：82-53-744-4301 **韩国 - 首尔** 电话：82-2-554-7200 **马来西亚 - 吉隆坡** 电话：60-3-7651-7906 **马来西亚 - 槟榔屿** 电话：60-4-227-8870 **菲律宾 - 马尼拉** 电话：63-2-634-9065 **新加坡** 电话：65-6334-8870 **泰国 - 曼谷** 电话：66-2-694-1351 **越南 - 胡志明市** 电话：84-28-5448-2100	**奥地利 - 韦尔斯** 电话：43-7242-2244-39 传真：43-7242-2244-393 **丹麦 - 哥本哈根** 电话：45-4450-2828 传真：45-4485-2829 **芬兰 - 埃斯波** 电话：358-9-4520-820 **法国 - 巴黎** 电话：33-1-69-53-63-20 传真：33-1-69-30-90-79 **德国 - 加兴** 电话：49-8931-9700 **德国 - 哈恩** 电话：49-2129-3766400 **德国 - 海尔布隆** 电话：49-7131-67-3636 **德国 - 卡尔斯鲁厄** 电话：49-721-625370 **德国 - 慕尼黑** 电话：49-89-627-144-0 传真：49-89-627-144-44 **德国 - 罗森海姆** 电话：49-8031-354-560 **以色列 - 赖阿南纳** 电话：972-9-744-7705 **意大利 - 米兰** 电话：39-0331-742611 传真：39-0331-466781 **意大利 - 帕多瓦** 电话：39-049-7625286 **荷兰 - 德卢内市** 电话：31-416-690399 传真：31-416-690340 **挪威 - 特隆赫姆** 电话：47-7288-4388 **波兰 - 华沙** 电话：48-22-3325737 **罗马尼亚 - 布加勒斯特** 电话：40-21-407-87-50 **西班牙 - 马德里** 电话：34-91-708-08-90 传真：34-91-708-08-91 **瑞典 - 哥德堡** 电话：46-31-704-60-40 **瑞典 - 斯德哥尔摩** 电话：46-8-5090-4654 **英国 - 沃金厄姆** 电话：44-118-921-5800 传真：44-118-921-5820