

开发人员的串行引导加载程序

作者: Pavel Lajsner, Pavel Krenek, Petr Gargulak

1 项目目标

开发人员的串行引导加载程序为用户提供在 Freescale 大多数微控制器上在线更新现有硬件的最简单方法。在线编程不能替代任何调试和开发工具，它仅可用作一个通过串行异步端口或 USB 对嵌入式系统进行重编程的简单选项。开发人员的串行引导加载程序所支持的微处理器包括 8 位系列 HC08 和 HCS08 以及 32 位系列的 ColdFire 和 Kinetis。全新 Kinetis 系列支持 K 系列和 L 系列。

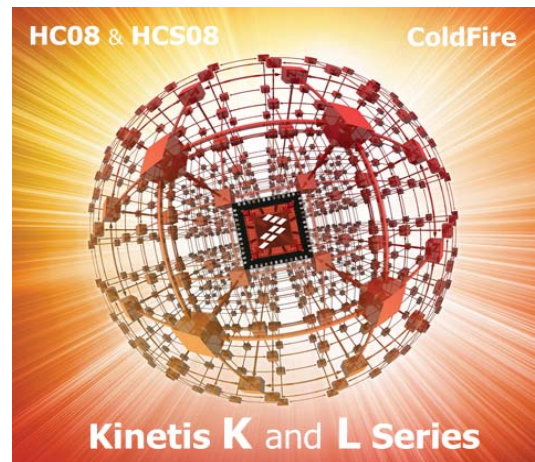
本应用笔记适用于关注替代性重编程工具的嵌入式软件开发人员。引导加载程序因其能在线修改 MCU 存储器而成为开发应用过程中可能非常有用的实用程序。

开发人员的串行引导加载程序对于演示用或最初用 MMDS 开发并需要做较小在线修改的应用来说是一种补充的实用程序。串行引导加载程序为已配备串行接口且 SCI 引脚外接至连接器的应用提供零成本解决方案。本文档还介绍其他编程技术：

- 使用 ROM 程序的 FLASH 重编程
- 简单的软件 SCI

目录

| | | |
|----|----------------------------|----|
| 1 | 项目目标 | 1 |
| 2 | FC 协议介绍 | 3 |
| 3 | FC 协议, 版本 1, M68HC908 实施 | 12 |
| 4 | FC 协议, 版本 2, HC9S08 实施 | 18 |
| 5 | FC 协议, 版本 3, 大 M68HC08 实施 | 22 |
| 6 | FC 协议, 版本 4, ColdFire (V1) | 22 |
| 7 | FC 协议, 版本 5, Kinetis | 29 |
| 8 | MCU 从机软件 | 36 |
| 9 | PC 引导加载程序主机软件 | 57 |
| 10 | 主机应用程序用户指南 | 62 |
| 11 | 合并引导加载程序 and 应用程序映像 | 70 |
| 12 | 参考文献 | 71 |



项目目标

- 适用于 USB（HC08JW、HCS08JM 和 MCF51JM MCU）的软件
- 内部时钟发生器的使用
- PLL 时钟编程
- EEPROM 编程（AS/AZ HC08 系列）
- CRC 串行协议保护选项

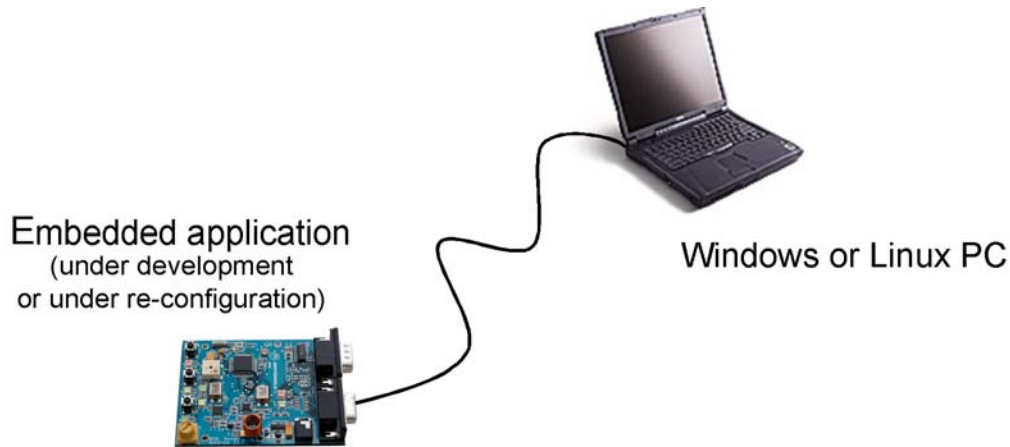


图 1. 顶层视图

1.1 项目目标

飞思卡尔半导体 M68HC08 MCU 使用标准监控模式接口进行 FLASH 编程。配置监控模式需要在 MCU 启动后将一个特定的时钟和高电压（进入监控模式的电压 $V_{TST} = V_{DD} + 2.5 = 8V$ ）外接到 IRQ 引脚上。此外，建立监控模式通信需要使用若干引脚。如果应用程序已使用标准串行 SCI 接口进行通信，那么重编程可以使用不同的代码（串行引导加载程序）通过相同的接口与 PC 通信。

引导加载程序仅可用于重编程，不可用于在线调试。引导加载程序是一个低成本的在线编程解决方案。

1.2 引导加载程序应用要求

以下几点介绍了引导加载程序应用的重要参数：

- **使用的存储空间少**— 引导加载程序必须使用尽可能少的存储空间。其他版本的引导加载程序使用的内存大于 1 KB，这对于可用内存为 3 KB 的器件（例如 MC68HC908JK3）来说是不可接受的。本文档中介绍的解决方案尽可能地简单实施除校验和等之外的所有特性。对于 8 位 MCU 来说，目标大小不到 500 B。USB 版本的引导加载程序包括适合于通过 USB 进行通信的驱动程序。因此，引导加载程序需要 8KB 的可用内存（HCS08JM 和 MCF51JM）。Kinetis K 系列和 L 系列的大小相似（小于 2 KB）。
- **低引脚数**— 该引导加载程序使用的是已实施的标准通信方式（通常板载 SCI 主要用于通信）。标准 SCI 使用两根线：RxD 和 TxD。无需额外用线即可启动引导加载程序。

- **有关用户 S19 文件的透明度** — 整个应用程序应当对用户代码 S19 文件透明。这意味着无需在 S19 文件中进行任何调整。其他 M68HC08、HCS08 和 ColdFire V1 引导加载程序应用需要修改中断向量或对 S19 文件作其他修改以使其接受引导加载程序。

1.3 引导加载程序应用的演示特性

本文档介绍 M68HC(S)08、ColdFire V1 (CFV1) 和 Kinetis 这几种不同引导加载程序的实施，它们主要因目标 M68HC(S)08、CFV1 和 Kinetis MCU 所具有的特性不同而各异。此外，还演示了 M68HC(S)08、CFV1 和 Kinetis 系列的若干特性，因此本文档非常有用，适用范围更广，不仅仅局限于只需要引导加载程序的受众。M68HC(S)08、CFV1 和 Kinetis 的不同实施还表现为以下特性：

- 使用内置 ROM 例程进行 FLASH 自编程（另请参见[参考文献](#)中的 AN1831、AN2545 和 AN2635）。
- 在 MC68HC908GP 系列或 MC9S08GB/GT 系列等无 ROM 的 MCU 上由用户实施在线重编程例程。
- 使用不同的 FLASH 模块保护技术实施（MC68HC908GP、MC68HC908GR、MC68HC908EY，与 MC68HC908JK/JL 系列相比）。
- 在 MC68HC908JK/JL 系列等无 SCI 的 MCU 上实施软件 SCI。
- 对于 HCS08 系列（MC9S08GB/GT），使用内部时钟发生器及其调整功能（适用于 MC68HC908KX 系列）。
- EEPROM 编程（适用于 MC68HC908AB/AS/AZ 系列）。
- 在 MC68HC908JW 系列、HCS08JM 和 MCF51JM 系列等 USB2.0 全速 HS08 MCU 上实施 USB 通信。
- 使用适用于 HCS08 和 ColdFire (V1) 器件的 Flash 编程实施（另请参见[参考文献](#)中的 AN3492）。

2 FC 协议介绍

如[引导加载程序应用要求](#)中所述，实施必须简单且尽可能少使用内存。因此，主机 PC 和从机 MCU 之间运行的协议也非常简单。之所以称之为 FC 协议，是因为使用了一个标志字符（应当或 ACK）0xFC 或 11111100b。

本节介绍对 MCU 进行重编程时 PC 和 MCU 之间通信所使用的协议。在简介之后会对特定于系列的实施特性加以说明。

下图为简单的状态图，显示本文档中介绍的引导加载程序的各种状态：

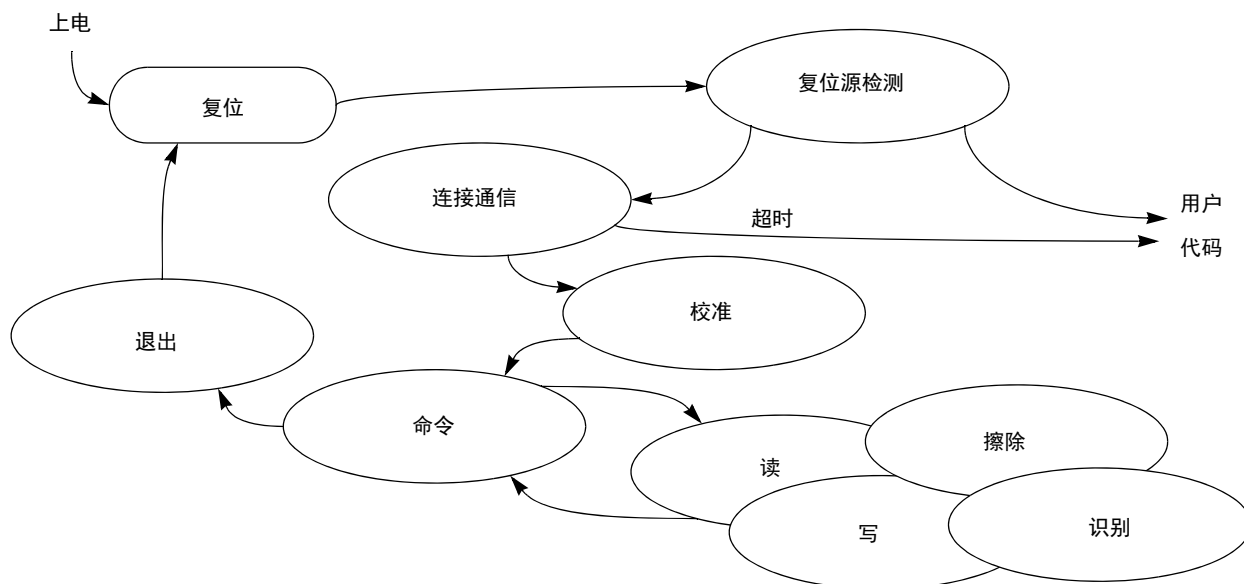


图 2. 引导加载程序的简化流程图

2.1 初始连接

可使用几种方法进入引导加载程序模式。其他一些解决方案使用“特定引脚采用特定电平”方法。例如，如果在 MCU 启动过程中 IRQ 引脚上出现逻辑 0，则引导加载程序代码启动，否则，用户代码会启动。

由于所使用的引脚数必须最少，因此开发人员的串行加载引导程序使用“特定时间采用特定字符”的方法。这意味着 MCU 通过串行接口发出 ACK 字符并等待响应。如果在指定时间内未接收到任何字符（连接超时），则该进程采用用户代码继续。

如果该情况出于任何原因而成为限制，则用户可修改引导加载程序代码以满足应用需求（例如，可在启动时另外实施简单的 IRQ 引脚检测）。更多详情，请参见 [M68HC08 系统限制](#)。

2.2 时钟源

FC 协议允许两种情形，具体取决于 MCU 是以已知的精确频率运行还是使用 RC（寄存器、电容）时钟或内部时钟（或任何编译时未知的时钟）。

2.2.1 未知 MCU 通信速度

如果频率不确定（编译时间未知），MCU 不会检查接收到的 ACK 字符是否仅符合 0xFC 模式。由于 MCU 时钟存在误差，某些字符的解读可能与 PC（图 3）发送的原始 0xFC 不同。对 MCU 侧进行的 0xFC 模式检查可以完全省略，这样可节省 MCU 内存。

PC 以适当的数据速率传送 0xFC 字符：

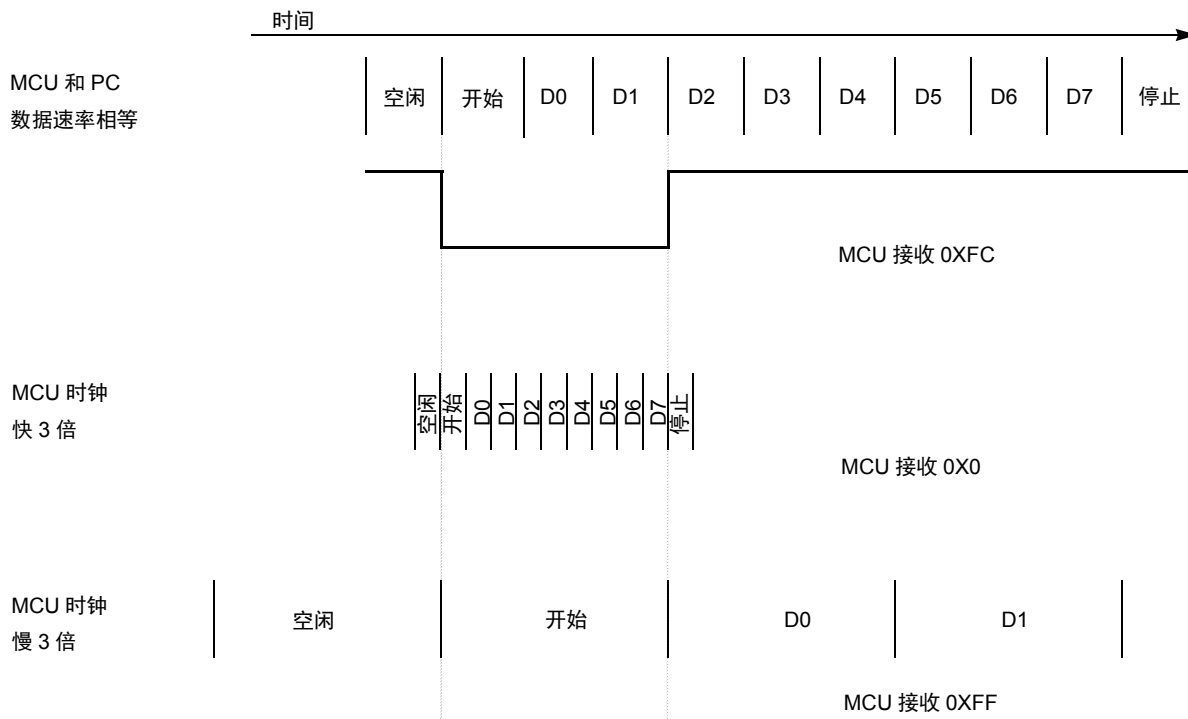


图 3. 匹配不同的通信速度

下表显示的是当传送速度和接收速度不相等时可正确接收（无成帧或噪声错误）的字符：

表 1. PC 至 MCU 传送 — 数据速率不匹配

| PC 数据速率 | MCU 数据速率 | 以二进制形式接收的字符 | 以十六进制形式接收的字符 |
|---------|-------------------|-------------|--------------|
| 9600 | $9600 \times 1/3$ | 11111111b | 0xFF |
| 9600 | $9600 \times 2/3$ | 11111110b | 0xFE |
| 9600 | $9600 \times 3/3$ | 11111100b | 0xFC |
| 9600 | $9600 \times 4/3$ | 11111000b | 0xF8 |
| 9600 | $9600 \times 5/3$ | 11110000b | 0xF0 |
| 9600 | $9600 \times 6/3$ | 11100000b | 0xE0 |
| 9600 | $9600 \times 7/3$ | 11000000b | 0xC0 |
| 9600 | $9600 \times 8/3$ | 10000000b | 0x80 |
| 9600 | $9600 \times 9/3$ | 00000000b | 0x00 |

如果 MCU 以不匹配的数据速率传送至 PC，则 PC 接收到的（和接受的）字符与 0xFC 字符不同。PC 接受来自括号中所述字符集（0xFF、0xFE、0xFC、0xF8、0xF0、0xE0、0xC0、0x80 和 0x00）的所有字符。如果已接收到字符，则 ACK 会被立即发送回 MCU。MCU 识别出该响应后，就会进入下一阶段：[从机频率校准](#)。

2.2.2 已知 MCU 通信速度

如果频率确定（已知编译时间），则 MCU 会被配置为与 PC 的通信速度完全匹配。所有字符均会被正确接收，并且不会产生任何失真。

MCU 将 0xFC 发送给 PC，而 PC 则立即向 MCU 发送 ACK。接收到 ACK 之后，MCU 也将（正式）进入从机频率校准阶段。

2.3 从机频率校准

在该阶段，MCU 时钟会被校准。至此，PC 与 MCU 之间的通信速率误差为 33% 至 300%。在该阶段，必须将 MCU 通信速度调整为与 PC 通信速度匹配。

PC 进入校准阶段后，无中断超时开始。如果在此期间未接收到正确的 ACK 字符 (0xFC)，则会以通信数据速率发送一个中断字符。

中断字符由 10 个连续的逻辑 0 组成。例如，如果波特率为 9600，则其高低高型脉冲持续时间为 $10 \times 104 \mu\text{s} = 1.04 \text{ ms}$ 。

随后，MCU 测量中断字符长度并确定其时钟是过快还是过慢。接着，MCU 会调整其系统时钟（或调整接收例程，例如使用软件串行通信时）。此过程可视需要重复多次，直至 MCU 达到正确的时钟速度。

附注：虚拟端口变通方案

大多数用户使用的是虚拟串行端口，但部分标准无法传输中断校准字符。因此，添加了使用零校准字符的全新特性，用以代替中断字符脉冲（图 4）。零校准字符由 9 个连续的逻辑 0 组成。

带 0 字符的校准特性在主应用程序中实施为“short TRIM”（复选框“short TRIM”，主机应用程序用户指南）。目标必须配置为使用短校准（调整）脉冲。

将 MCU 校准为正确时钟（或校准接收例程）后，ACK 字符会被发送给 PC，以停止发送校准字符（图 4）。

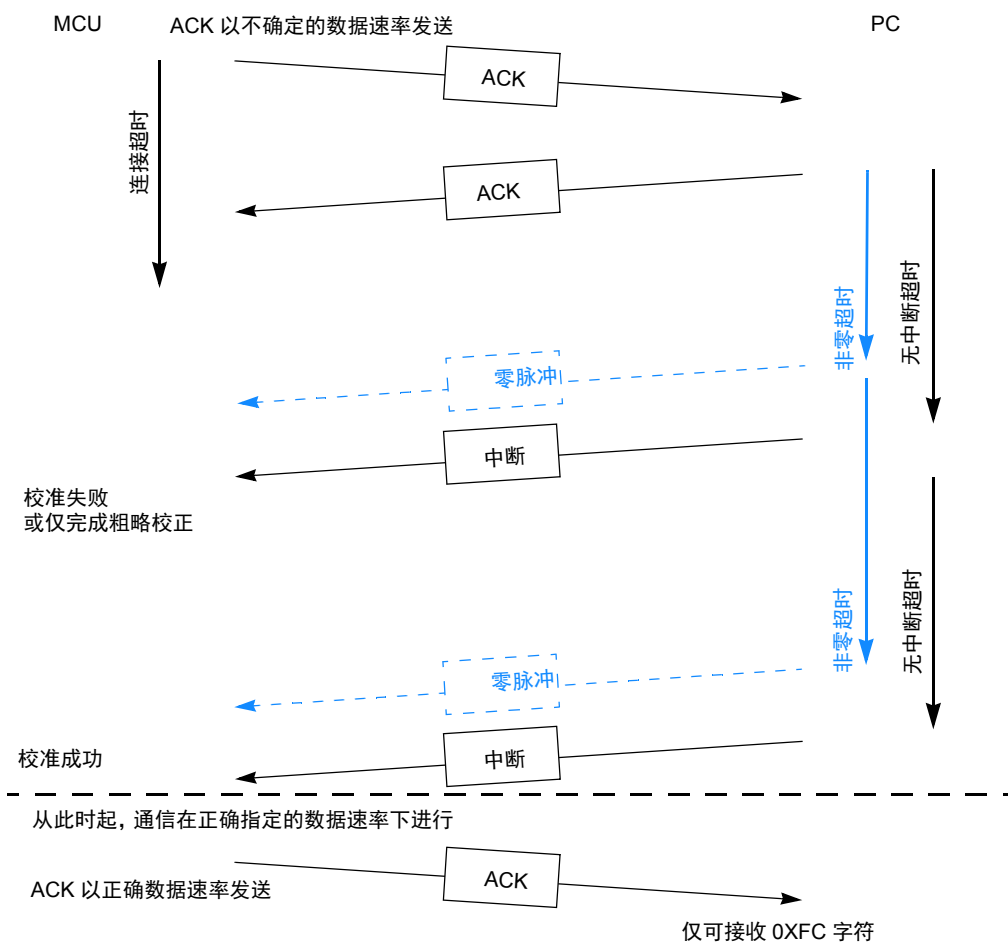


图 4. 启动通信（经过校准）

如果 MCU 以正确的数据速率工作（无法校准或无需校准，且 MCU 时钟由晶体驱动），则 PC 可立即发送 ACK，完全跳过校准阶段（图 2）。

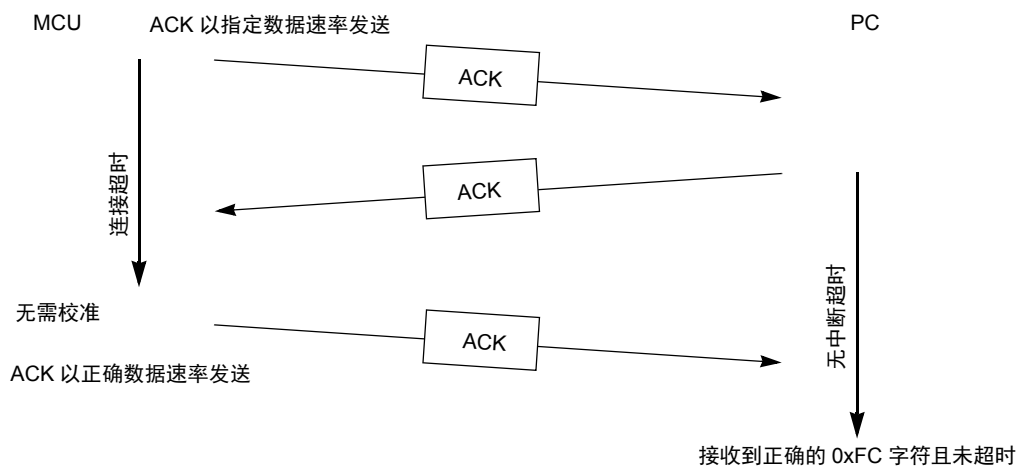


图 5. 启动通信（未校准）

2.4 解释 MCU 命令

MCU 与 PC 之间的通信建立后，MCU 会进入主命令解释器循环。MCU 执行简单的命令，以便对其自有的非易失性存储器进行重编程。通信将按照主从机制执行：PC 发出命令，MCU 执行命令并通过数据或单个 ACK 字符确认已完成每条命令。

最小命令集包括：

- Ident 命令
- 退出命令

对于纯粹的重编程，需要另外实施两个基本命令：

- 擦除命令
- 写命令

如果用户需要验证特性，则必须将额外的（读取）命令编译到 MCU 代码中。对于纯粹的重编程（最小配置），无需实施此命令。

- 读命令

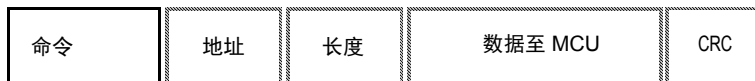
CRC 安全协议实施

该协议可为所有消息的 CRC 安全提供选择。CRC 使用标准 16 位实施 CCITT16 且将 0xFFFF 用作复位值。

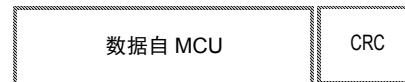
擦除命令示例值：

```
'E'-1byte - 0x45
'start address' - 2 bytes - 0x1234
'CRC - 2 bytes' - 0x2907
```

PC 至 MCU 命令



MCU 至 PC 响应



* 虚线部分字段并非总会实现，相反，来自 MCU 的数据可能只包含 ACK 字符。

图 6. 典型的命令和响应

2.4.1 Ident 命令

ident 命令（代码为 'I', \$49）没有附加域。

该命令由 PC 在通信建立之后立即发出。ident 命令旨在将正在编程的 MCU 的一些基本属性通知 PC。发送所有多字节字段时均以最高有效位开始。

- 版本号和功能表 - 1 个字节

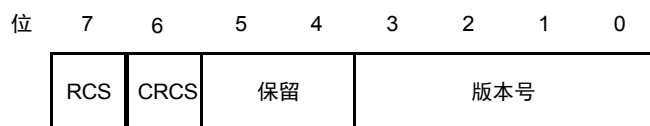


图 7. 版本号和功能表

- RCS - “支持读命令 (RCS)” 标志会通知 PC 是否支持（实施）读命令。如果不支持，则 MCU 会忽略读例程的所有调用，并且不会向 PC 发送任何响应。PC 软件会提醒用户读功能不可用。
 - 支持 - 不支持（通常是出于内存约束）
- CRCS - “支持 CRC 串行协议” 标志会通知 PC 所有其他通信（包括 Ident 命令）是否已由 CRC-CCITT 校验和保证安全。
 - 支持¹ - 不支持（通常是出于内存约束）
- RSVD - 这些位将为后续使用、闲置而保留，且应置 0。
- VER — 协议版本

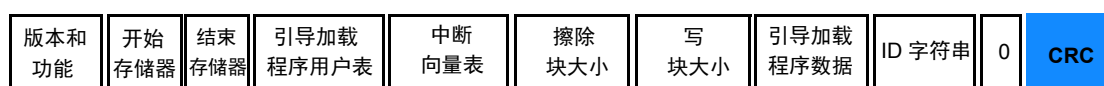
2.4.2 FC 协议版本 1 (M68HC08)

协议版本 1 适合于 M68HC08 MCU。版本 1 中其他字段的定义如下：

- 可重编程内存区的起始地址 - 2 个字节。
- 可重编程内存区的结束地址 + 1 - 2 个字节。
- [引导加载程序用户表](#)的地址 - 2 个字节。
- MCU 中断向量表的起始地址 - 2 个字节。
- MCU 擦除块的长度 - 2 个字节。
- MCU 写块的大小 - 2 个字节。
- 引导加载程序数据（特定引导加载程序信息，请参见特定于器件的实施；在 [表 2](#) 中对比）- 8 个字节。
- 以零结尾的识别字符串 - <n> 个字节。
- 如果已启用串行协议的 CRC 功能，则后面为 CRC-CCITT 校验和 - 2 个字节。

PC 至 MCU 命令

I (\$49)



MCU 至 PC 响应

图 8. Ident 命令（FC 协议版本 1，M68HC08）

1. 自 2011 年第 3 季度起提供

2.4.3 FC 协议版本 2 (HCS08) 和 FC 协议版本 3 (大 M68HC08)

协议版本 2 适合于 HCS08 MCU；协议版本 3 适合于大 M68HC08（具有两个或以上 FLASH 存储库的 HC08）。对于两种版本，其他字段的定义如下：

- 系统器件识别寄存器内容 — 2 个字节（不用于协议版本 3，代码为 \$FFFF）
- 可重编程内存区的数目 (N) - 1 个字节
- 可重编程内存区 1 的起始地址 - 2 个字节
- 可重编程内存区 1 的结束地址 + 1 - 2 个字节
- 可重编程内存区 2 的起始地址 - 2 个字节
- 可重编程内存区 2 的结束地址 + 1 - 2 个字节
- 可重编程内存区 N 的起始地址 - 2 个字节
- 可重编程内存区 N 的结束地址 + 1 - 2 个字节
- 重定位中断向量表的地址 - 2 个字节
- MCU 中断向量表的起始地址 - 2 个字节
- MCU 擦除块的长度 - 2 个字节
- MC 写块的长度 - 2 个字节
- 以零结尾的识别字符串 - <n> 个字节
- 如果已启用串行协议的 CRC 功能，则后面为 CRC-CCITT 校验和 - 2 个字节

PC 至 MCU 命令

I (\$49)

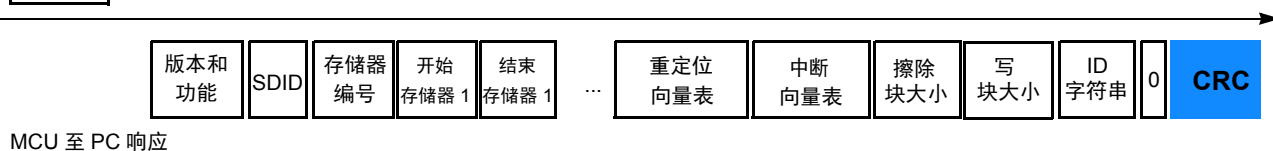


图 9. Ident 命令（FC 协议版本 2 和 3，HCS08）

2.4.4 擦除命令

擦除命令（代码为 ‘E’, \$45）仅具有地址字段，不存在长度字段或数据字段。起始地址为 2 字节字段，从最高有效位开始。如果已启用串行协议的 CRC 功能，则 16 位（2 个字节）后为 CRC-CCITT 校验和。

存在指定地址时，MCU 会擦除地址块。要擦除的地址块的长度等于擦除块大小（通常取决于硬件）。

MCU 执行完命令后，ACK (\$FC) 字符会被发送给 PC。如果已启用串行协议的 CRC 功能，则 16 位（2 个字节）后为 CRC-CCITT 校验和。未指定擦除命令的最小和最大执行次数。



图 10. 擦除命令

2.4.5 写命令

写命令（代码为 ‘W’, \$57）包含地址字段和数据字段。该地址包含要编程的起始地址。起始字节是后跟要编程的字节数的长度。起始地址是 2 字节字段，以最高有效位开始，长度则是 1 字节字段。如果已启用串行协议的 CRC 功能，则 16 位（2 个字节）后为 CRC-CCITT 校验和。

MCU 执行完命令后，ACK (\$FC) 字符会被发送给 PC。如果已启用串行协议的 CRC 功能，则 16 位（2 个字节）后为 CRC-CCITT 校验和。未指定写命令的最小和最大执行次数。

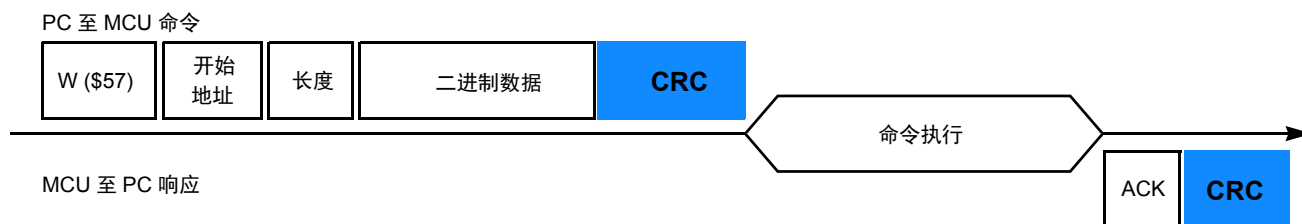


图 11. 写命令

2.4.6 读命令

读命令（代码为 ‘R’, \$52）包含地址字段和数据字段。该地址包含要编程的起始地址；单个字节为要读取的数据的长度。起始地址是 2 字节字段，以最高有效位开始，长度则是 1 字节字段。如果已启用串行协议的 CRC 功能，则 16 位（2 个字节）后为 CRC-CCITT 校验和。

MCU 将该读取字节数发送至 PC。如果已启用串行协议的 CRC 功能，则 16 位（2 个字节）后为 CRC-CCITT 校验和。

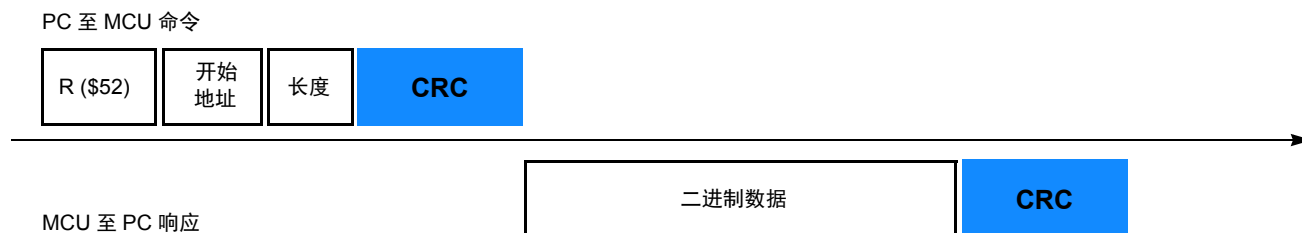


图 12. 读命令

2.4.7 退出命令

读命令（代码为‘Q’，\$51）没有地址字段和数据字段。引导加载程序执行会立即完成，并且用户代码会启动。不会向 PC 发送任何 ACK (\$FC) 字符。

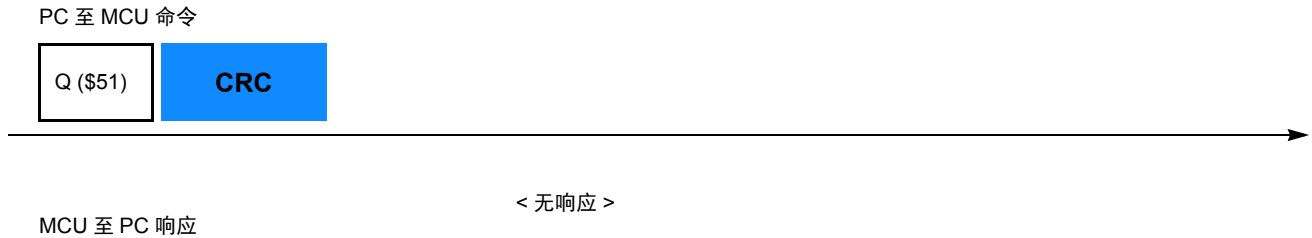


图 13. 退出命令

2.4.8 引导加载程序用户表

引导加载程序用户表是一个可重编程内存区，用于存储引导加载程序特定数据。该内存区对于用户程序不可用。有关该表的内存分配，请参阅 [FC 协议，版本 1，M68HC908 实施](#)。

3 FC 协议，版本 1，M68HC908 实施

本节介绍针对于 M68HC908 引导加载程序实施的功能。内存分配高度取决于 MCU，在本节中会详细介绍所有变量的含义。

图 14 介绍引导加载程序经过预编程的 M68HC908 MCU 的典型内存分配。例如，MC68HC908KX8 MCU 内存映射包括：

- 7680 字节 FLASH 存储器 (\$E000–\$FDFF)
- 192 字节随机访问存储器 (RAM) (\$0040–\$00FF)
- 36 字节用户定义向量 (\$FFDC–\$FFFF)

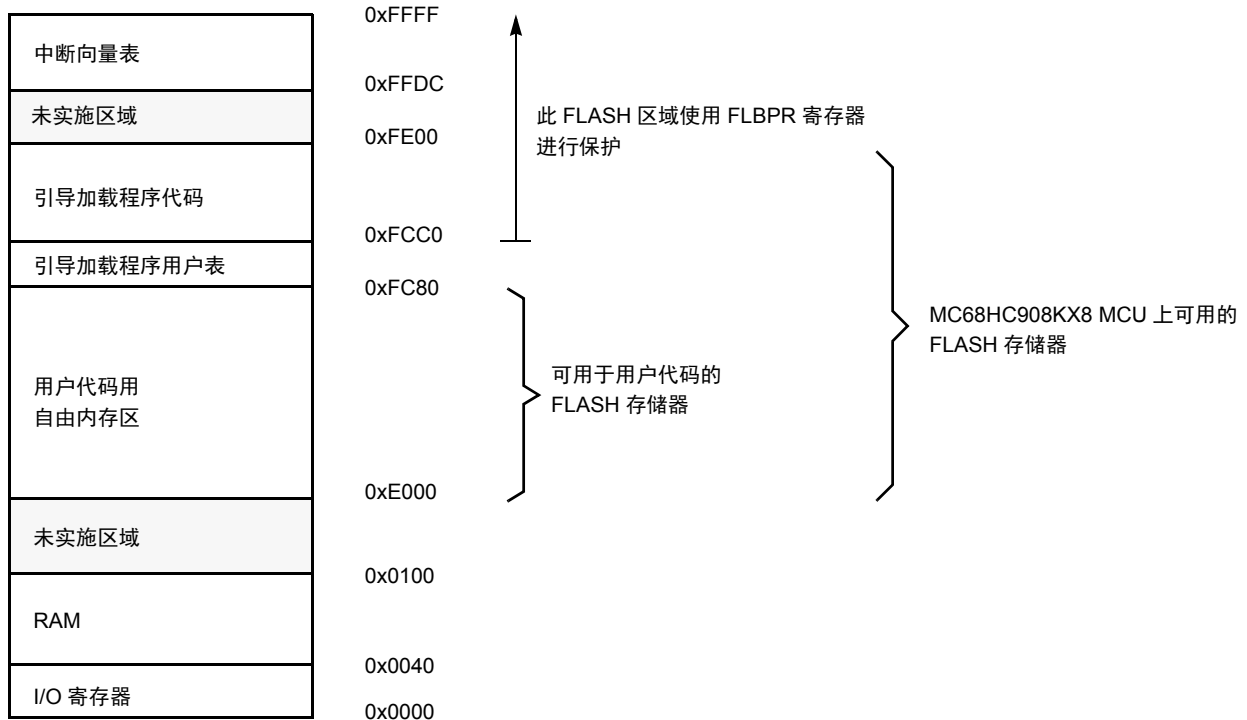


图 14. MC68HC908KX8 内存分配简单实例

3.1 内存分配

引导加载程序代码占用 FLASH 存储器顶端（最高内存地址空间）。这种布置能够有效利用 FLASH 的块保护技术（详情请参见特定的 MCU 数据手册）。

3.2 FLASH 块保护寄存器 (FLBPR)

通过设置 FLBPR（FLASH 块保护寄存器），该地址以上的所有地址空间都会受到保护，不会被有意和无意的擦除 / 重新写入。引导加载程序和 FLBPR 寄存器均被编入到存储器中，引导加载程序代码会受到保护，不会被用户代码无意修改。

附注

某些 M68HC908 MCU 的 FLBPR 寄存器位于 RAM 而不是 FLASH 中（例如，MC68HC908JK/JL 系列）。引导加载程序可正确设置此寄存器，但用户代码可完全修改此 FLBPR，并且可以最终修改和擦除 / 写入引导加载程序代码。请参见 [FLBPR 不可用（在某些 M68HC08 系列 MCU 中）](#)。

例如，MC68HC908KX8 引导加载程序至 PC 的内存分配为：

- \$01 - 版本 1，读命令未实施（位 7）。
- \$E000 - 可重编程内存区的起始地址。
- \$FC80 - 可重编程内存区的结束地址 + 1。
- \$FC80 - [引导加载程序用户表](#)的地址。

- \$FFDC - MCU 中断向量表的起始地址。
- \$0040 - MCU 擦除块的长度。
- \$0020 - MCU 写块的长度。
- 0,0,0,0,0,0,0 - 引导加载程序数据。未严格定义语法；不同的 M68HC08 实施可提供不同的值（例如，MC68HC908KX8 实施中的第六个值为校准后的内部时钟发生器 [ICG] 调整寄存器的值）。所有这些引导加载程序数据随后都会被编程回引导加载程序用户表中，并且可以在所有后续启动过程中进行检索（例如，在用户代码启动之前，将 MCU 的 ICG 调整至已知的最佳值）。
- ‘KX8-IR’,0 - 以零结尾的识别字符串。将显示在 PC 屏幕上的信息。

3.3 中断向量表重定位

由于 FLASH 块保护技术还可保护中断向量表使其免受覆盖，因此必须采用某种方法将这些向量重定位至不同位置。为此，需使用引导加载程序用户表。该表是未受 FLBPR 保护的存储器的一部分，但它不可用于用户程序。所有标准中断向量均指向该表，每个中断的 JMP 指令均存储在该表中。唯一例外的是复位向量指向的是引导加载程序代码的开始处。发生中断时，向量从受保护的存储器中被提取出来，并根据引导加载程序用户表中相应的 JMP 指令引导执行继续。

下图说明了 M68HC08 MCU 的中断向量表重定位。

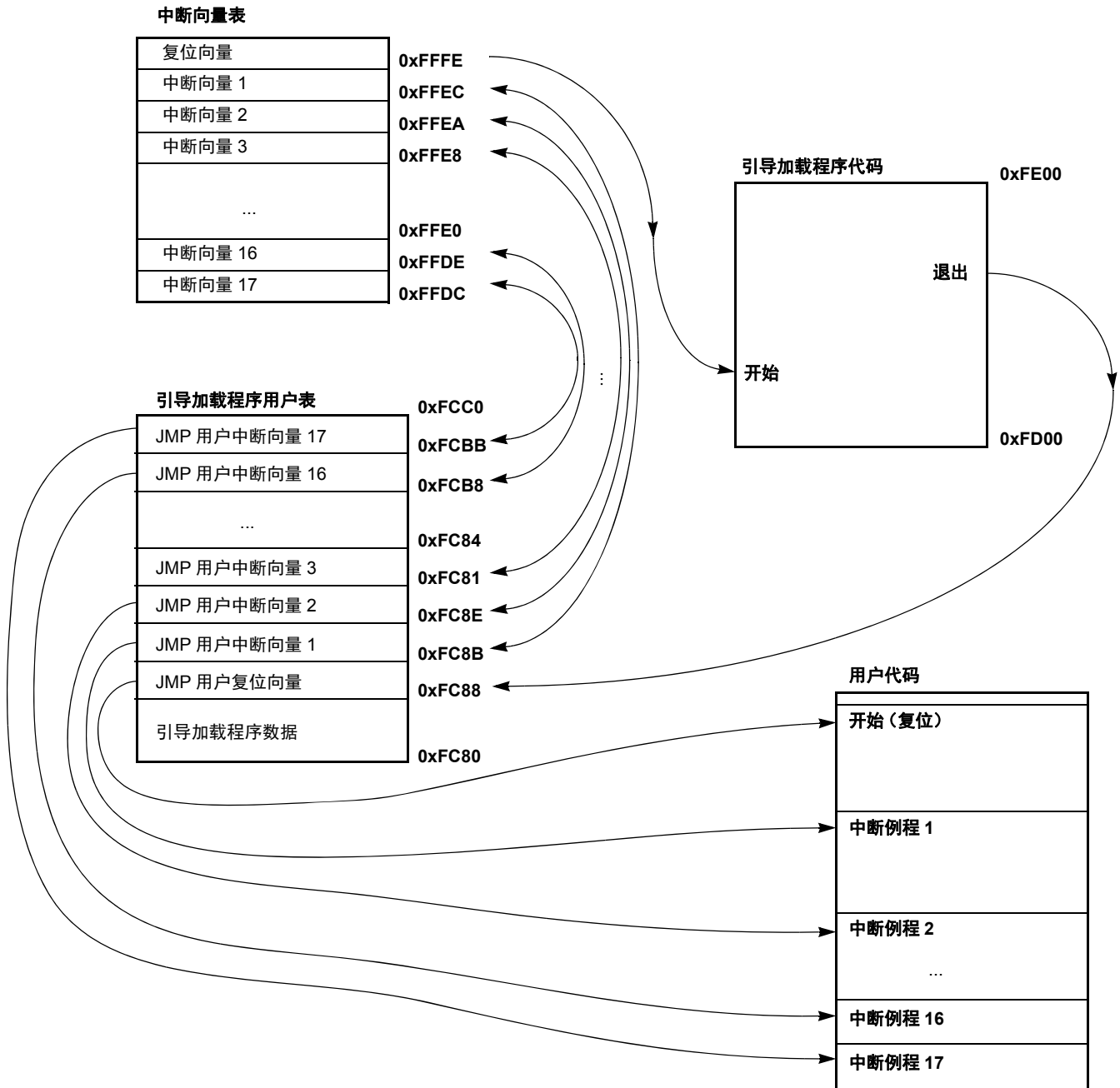


图 15. 中断向量表重定位 (M68HC08 MCU)

附注

在标准中断向量表中, 每条记录的长度为 2 个字节 (每个向量为一个 16 位地址)。这不同于引导加载程序用户表, 对于后者, 每条记录的长度为 3 个字节 - 一个 JMP 操作码 (SCC) + 一个 16 位地址。

3.3.1 S19 文件

由于引导加载程序的操作必须对用户 S19 文件保持透明, 因此, 在 PC 主机代码 (而不是 MCU 从机代码) 中置入了另一条信息。重定位工作如下:

如果 S19 记录中的数据对应于中断向量表中的地址, 则该值会被重定位至引导加载程序用户表中的相应区域, 包括 JMP 指令 (操作代码 \$CC)。例如, 如果用户 S19 文件的地址 \$FFE8 包含 #3 中断向量 \$E123, 则此类向量将被重定位到已编程至引导加载程序用户表中 \$FC81 地址的序列 SCC、\$E1、\$23 (JMP \$E123)。

通过此方法, 无需修改用户 S19 文件, 但必须考虑 FLASH 存储器末尾的低位地址。此外, 该 JMP 指令 (3T) 在每个中断时都会延迟, 如[每个中断 3T 延迟](#)中所述。

3.4 用户代码启动

用户代码以不寻常的方式启动, 使寄存器设置与其在 MCU 复位后的设置相似。

3.4.1 软件复位

如果引导加载程序必须退出并运行用户代码, 则需要有意执行非法操作 (M68HC08 非法操作码 \$32)。这会导致非法操作复位且 MCU 会重启。在引导加载程序启动过程中, 会检测系统集成模块 (SIM) 复位状态寄存器 (SRSR)。如果未检测到上电复位, 则启动的会是用户代码, 而非引导加载程序代码。这就允许透明操作所有其他复位 (例如非法地址等), 并且检测 SRS 寄存器和执行关联的跳跃指令只会引起短时额外延迟。

3.4.2 硬件复位

在某些实施中, 引脚复位 (由外部复位引脚引起) 也是引导加载程序启动的有效复位源。这就允许在能够驱动 M68HC08 复位引脚的嵌入式应用中进行远程在线重编程。

实际引导加载程序应用中增加了另外一个检测: 如果未检测到复位源 (即, 如果 SRS 寄存器为 0), 则会默认选择引导加载程序。当外部引脚引起复位时可能会发生这种情况, 但复位脉冲短于指定值。在该情况下, 引起复位的复位脉冲的最小长度短于外部复位标志正确传播至 SRS 寄存器所需的长度。

由于 SRS 寄存器只可读取一次 (读取后会被清零), 因此, 后续对该寄存器进行读操作将不会提供有效值。详情请参见 [M68HC08 系统限制](#)。

3.5 M68HC08 系统限制

本节总结将引导加载程序与用户应用程序结合使用时必须考虑的限制情况。

3.5.1 占用的内存

其中一个最重要的要求是尽可能使用最小的代码。典型的 M68HC908 实施在 300 至 500 个字节之间, 包括引导加载程序用户表。如果目标 M68HC08 MCU 能够使用内部 ROM 例程进行 FLASH 编程, 那么内存消耗就接近下限值。大 M68HC08 MCU (通常未配备用于 FLASH 编程的 ROM 代码) 大约需要消耗 FLASH 总共 32 KB 中的 500 个字节 (MC68HC908GP32 也一样)。

引导加载程序位于 FLASH 存储器顶端, 因此, 用户代码中所需的唯一修改为内存映射 (通常可以在链接器参数文件中找到)。

M68HC08 MCU 发送实际可用 FLASH 地址的信号。如果用户代码与引导加载程序代码重叠, 则 PC 引导加载程序软件将不会允许编程。

3.5.2 启动和初始化通信时的延时

引导加载程序启动期间具有特定含义的引脚的数目必须尽可能的少。尤其是在通信系统中 (例如: 使用标准串行端口的系统), 引脚开销为零且使用的是“*特定时间采用特定电平字符*”的方法。因此, 引导加载程序会在启动时等待一定的时间, 以接收来自 PC 的响应。如果未接收到任何响应, 用户代码会启动。典型的延迟范围为几百毫秒。

如果该启动延迟对最终应用来说成为一个问题, 则用户可以修改引导加载程序代码并使用“*特定引脚采用特定电平*”的方法。可以通过简单检测 IRQ 引脚 (或任何其他输入引脚) 上的电压电平来确定是否需要引导加载序列。

3.5.3 每个中断 3T 延迟

每次中断调用都会延迟 3T 总线时钟, 这是执行存储在引导加载程序用户表中的 JMP 指令所需的。这种中断向量重定位方法 (如[中断向量表重定位](#)中所述) 已被选为实现用户代码透明性以及引导加载程序代码安全性的最佳解决方案。

中断延迟约为 10-15T (假定此时未执行任何中断), 因此, 这一额外延迟对于大多数应用来说影响不大。

3.5.4 FLBPR 不可用 (在某些 M68HC08 系列 MCU 中)

引导加载程序使用 FLASH 块保护技术保护其自身不被覆盖 (如适用; 详情请参见 [FLASH 块保护寄存器 \(FLBPR\)](#))。

某些 M68HC08 MCU (例如 KX、GP 和 GR 器件) 将该 FLASH 块保护寄存器存储在 FLASH 中, 使其在用户模式下无法被修改。只能通过 IRQ 引脚上存在的外部电压 V_{TST} 擦除 FLBP 或对其进行编程。由于该功能为引导加载程序代码保护专用, 因此, 它无法用于用户应用程序代码。如果 FLBPR 值出现在用户 S19 代码中, 则会显示一则警告。用户 S19 代码应该忽略这种情况的发生。

某些系列将 FLASH 块保护寄存器存储在 RAM 中 (例如 MC68HC908JK/JL 系列)。引导加载程序在开始执行自保护时设置正确的值。但是, 用户代码可以修改该寄存器并在需要时保护其自有的内存区。也就是说, 引导加载程序并非完全受用户代码保护。

有关详细说明, 请参见特定的 MCU 数据手册。

3.5.5 SRS 寄存器不可用

引导加载程序使用 SRS 寄存器 (如[用户代码启动](#)中所述) 识别复位源, 以确定用户代码是否运行。由于 SRS 寄存器只可读取一次 (即: 在复位后第一次读取), 因此, 用户代码无法访问 SRS 寄存器值 (如果引导加载程序存在于内存中且在每次复位后都进行第一次读取)。对于这种情况, 没有简单的解决方案。SRS 寄存器由引导加载程序读取后, 会被存储在某个 RAM 位置。其内存

位置可能因实施不同而各异。如果该应用需要 SRS 寄存器和引导加载程序，那么用户必须将 SRSR 读数重定向至此特定的 RAM 位置。该位置可从引导加载程序的 MAP 文件中获得。

4 FC 协议，版本 2，HC9S08 实施

本节介绍特定于 HC9S08 引导加载程序实施的功能。内存分配高度取决于 MCU，因此，在本节中会介绍变量的含义。

图 16 介绍引导加载程序经过预编程的 HC9S08 器件的典型内存分配。例如，MC9S08GB/GT60 器件的内存映射包括：

- 60 KB FLASH 存储器 (\$1080-\$17FF, \$182C-\$FFAF)
- 4 KB 随机访问存储器 (RAM) (\$0080-\$107F)
- 16 字节非易失性寄存器 (\$FFB0-\$FFBF)
- 64 字节用户定义向量 (\$FFC0-\$FFFF)

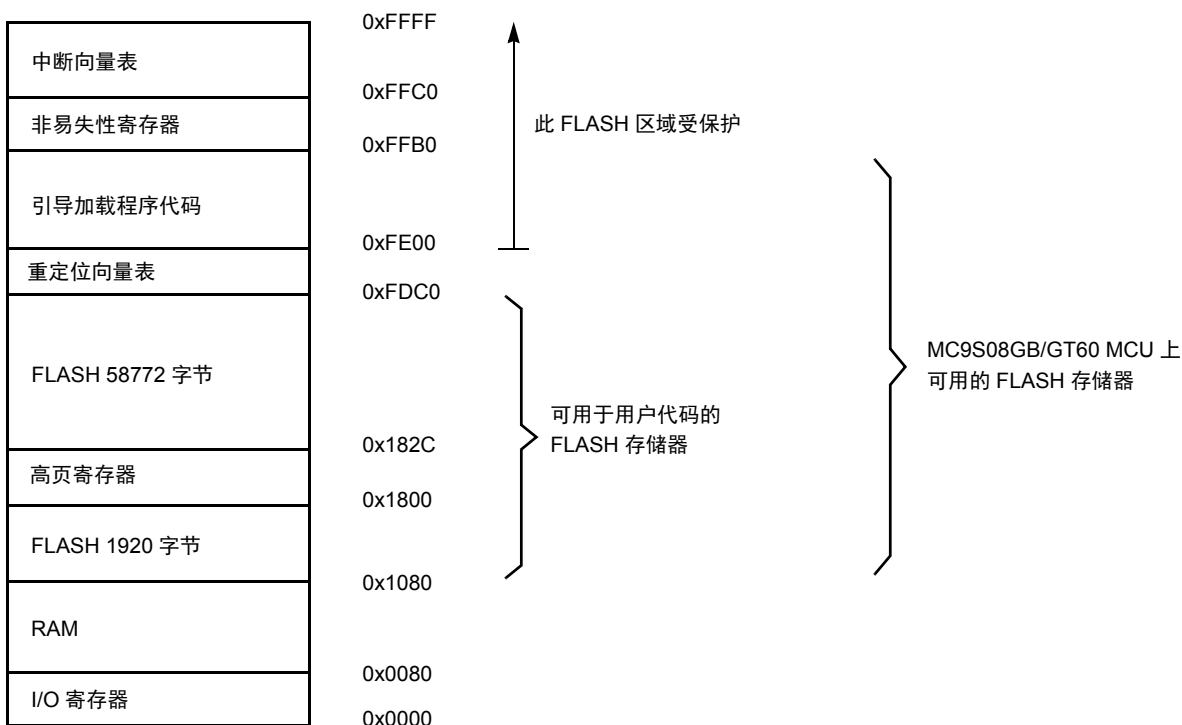


图 16. MC9S08GB/GT60 内存分配简单实例

4.1 内存分配

引导加载程序代码占用 FLASH 存储器顶端（最高内存地址空间）。这种设置能够有效利用 FLASH 保护技术（详情请参见特定的 MCU 数据手册）。

4.2 FLASH 保护

通过设置 FLASH 保护寄存器, 该地址以上的所有地址空间都将受到保护, 无法有意或无意擦除 / 重新写入。在引导加载程序和 FLASH 保护寄存器均被写入存储器中后, 引导加载程序会受到保护, 不会被用户代码意外改写。

附注

有关限制情况, 请参见 [FLASH 保护技术不可用](#)。

4.3 内存分配实例

MC9S08GB/GT60 引导加载程序至 PC 的内存分配实例如下:

- \$82 - 版本 2, 读命令已实施 (位 7)
- \$r002 - 系统器件识别寄存器 (SDIDR) 内容 (\$002 适用于 GB/GT 系列, r (高四位)) 为反映当前硅等级的芯片修订编号
- \$02 - 可重编程存储区的数目
- \$1080 - 可重编程内存区 1 的起始地址
- \$1800 - 可重编程内存区 1 的结束地址 + 1
- \$182C - 可重编程内存区 2 的起始地址
- \$FDC0 - 可重编程内存区 2 的结束地址 + 1
- \$FDC0 - 重定位中断向量表的地址
- \$FFC0 - MCU 中断向量表的起始地址
- \$0020 - MCU 擦除块的长度
- \$0040 - MCU 写块的长度
- 'GB/GT60',0 - 以零结尾的识别字符串。将显示在 PC 屏幕上的信息

4.4 中断向量表重定位

如果已启用 Flash 保护, 则复位和中断向量会受保护。向量重定向 (HCS08 硬件功能) 使用户能够修改中断向量信息的内存分配。

通过对 NVOPT (非易失性选项) 寄存器进行编程可启用向量重定向。为了使能重定向, 必须通过对 NVPROT (非易失性保护) 寄存器进行编程使至少一部分但并非全部的 FLASH 存储器受到块保护。除复位向量 (\$FFFE:FFFF) 外的所有中断向量 (存储器位置 \$FFC0-\$FFFD) 都会被重定向。

例如, 如果 512 字节的 FLASH 受保护, 那么受保护的地址区域为 \$FE00 至 \$FFFF。中断向量 (\$FFC0-\$FFFD) 会被重定向至位置 \$FDC0-\$FDFD。

例如, 如果发生 SPI 中断, 那么向量使用位置 \$FDE0:FDE1 中的值而非位置 \$FFE0:FFE1 中的值。这就使用户能在离开包括未更改默认向量位置在内的受保护区域的同时, 使用包括新中断向量值在内的新程序代码对 FLASH 的未受保护部分进行编程。

4.4.1 S19 文件

由于引导加载程序的操作必须对用户 S19 文件保持透明，因此在 PC 主机代码（而不是 MCU 从机代码）中置入了另一种信息。如果在用户 S19 文件中检测到中断向量表中的记录，则向量会被重定位至中断向量表中的相应区域。例如，如果用户 S19 文件的地址 \$FFEA 包含 #2 中断向量，则此类向量会被重定位至重定位中断向量表中的 \$FDEA 地址。

通过此方法，无需修改用户 S19 文件，但必须考虑 FLASH 存储器末尾的低位地址。

下图说明了 HC9S08 中断向量表的重定位：

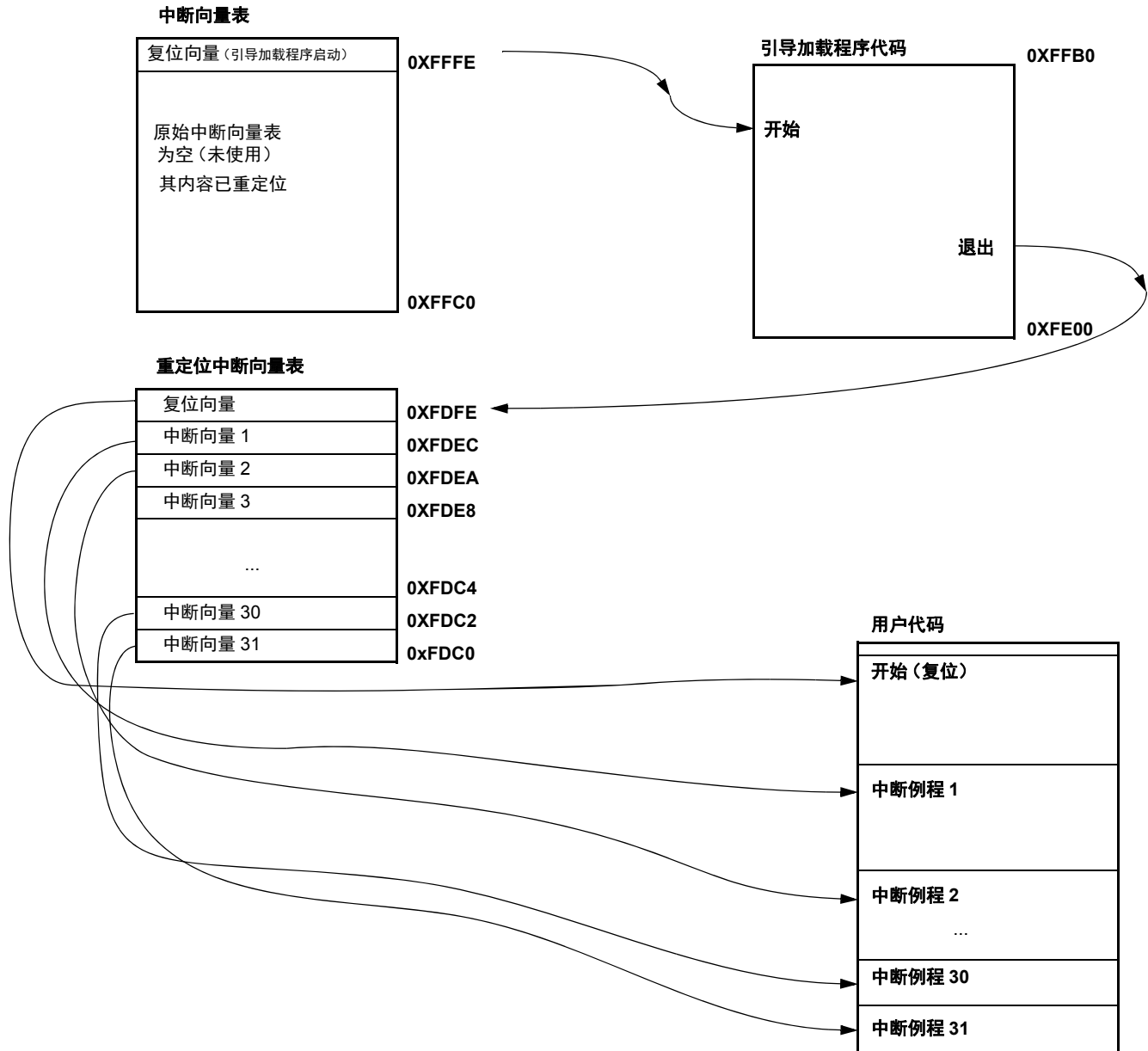


图 17. 中断向量表重定位说明 (HCS08)

4.5 用户代码启动

要使寄存器设置与其在 MCU 复位后的设置相似, 用户代码需以不寻常的方式启动。

4.5.1 软件复位

如果引导加载程序必须退出并运行用户代码, 则需要有意执行非法操作 (HCS08 非法操作码 \$8D)。这会导致非法操作复位且 MCU 会重新启动。在引导加载程序启动过程中, 会检测“系统复位状态 (SRS)”寄存器。如果未检测到上电复位, 那么启动的会是用户代码, 而非引导加载程序代码。这样就允许透明操作所有其他复位 (例如非法地址等), 并且检测 SRS 寄存器和执行关联的跳跃指令只会引起短时额外延迟。

4.5.2 硬件复位

在某些实施中, 复位引脚 (由外部复位引脚引起) 也是引导加载程序启动的有效复位源。这就允许在能够驱动 HCS08 MCU 复位引脚的嵌入式应用中进行远程在线重编程。

4.6 HCS08 系统限制

本节总结将引导加载程序与用户应用程序结合使用时必须考虑的限制情况。

4.6.1 占用的内存

其中一个最重要的要求是尽可能使用最小的代码。典型的 HC9S08 实施为 432 个字节 (可保护的最低内存大小) 加上另外 64 字节页 (用于重定位中断向量表)。

引导加载程序位于 FLASH 存储器的顶端, 因此, 用户代码中所需的唯一修改为内存映射 (通常可以在链接器参数文件中找到)。

HCS08 MCU 发送实际可用 FLASH 地址的信号。如果用户代码与引导加载程序代码重叠, 则 PC 引导加载程序软件在编程之前会发出警告。

4.6.2 启动和初始通信时的延时

引导加载程序启动期间具有特定含义的引脚的数目必须尽可能的少。尤其是在通信系统中 (例如, 使用标准串行端口的系统), 引脚开销为零且使用的是“特定时间采用特定字符”的方法。因此, 引导加载程序会在启动时等待一定的时间, 以接收来自 PC 的响应。如果未接收到任何响应, 用户代码会启动。典型的延迟范围为几百毫秒。

如果该启动延迟对最终应用来说成为一个问题, 则用户可以修改引导加载程序代码并使用“特定引脚采用特定电平”的方法。可以通过简单检测 IRQ 引脚 (或任何其他输入引脚) 上的电压电平来决定是否需要引导加载序列。

4.6.3 FLASH 保护技术不可用

引导加载程序使用 FLASH 块保护技术保护其自身不被覆盖，因此，该功能不可用于用户代码。这包括用于由引导加载程序执行的保护和中断向量重定位的 FLASH 存储器安全相关寄存器（即 NVPROT、NVOPT 和 NVBACKKEY）。

5 FC 协议，版本 3，大 M68HC08 实施

本节介绍针对于引导加载程序协议版本 3 的功能。该协议版本专用于大 HC08（具有两个或两个以上 FLASH 存储体，更准确地说是，具有两个或两个以上单独的 FLASH 内存区）。使用版本 2 中的 `Ident` 命令格式；其他保持与协议版本 1 (HC08) 相同，即中断向量表重定位。

6 FC 协议，版本 4，ColdFire (V1)

协议版本 4 分为两个版本：版本 A 和版本 B。这种划分主要根据对引导加载程序源代码保护的可能性来实施的。该功能对于 Flash 编程非常重要，因为保护引导加载程序可防止源代码被擦除。版本 A 引导加载程序的实施不受保护，而版本 B 的实施受保护。

6.1 版本 A（无保护版）

本节介绍针对于 Cold Fire V1 实施版本 A 的功能。内存分配为 MCU 特定，因此所有变量的含义都会在本节中进行说明。

图 18 介绍引导加载程序经过预编程的 ColdFire V1 器件的典型内存分配。例如，MCF51JM128 器件的内存映射包括：

- 128 KB FLASH 存储器 (\$00000000-\$0001FFFF)
- 16 KB 随机访问存储器 (RAM) (\$00800000-\$00803FFF)
- 16 字节易失性寄存器 (\$00000400-\$0000040F)
- 444 字节用户定义向量 (\$00000000-\$000001B8)

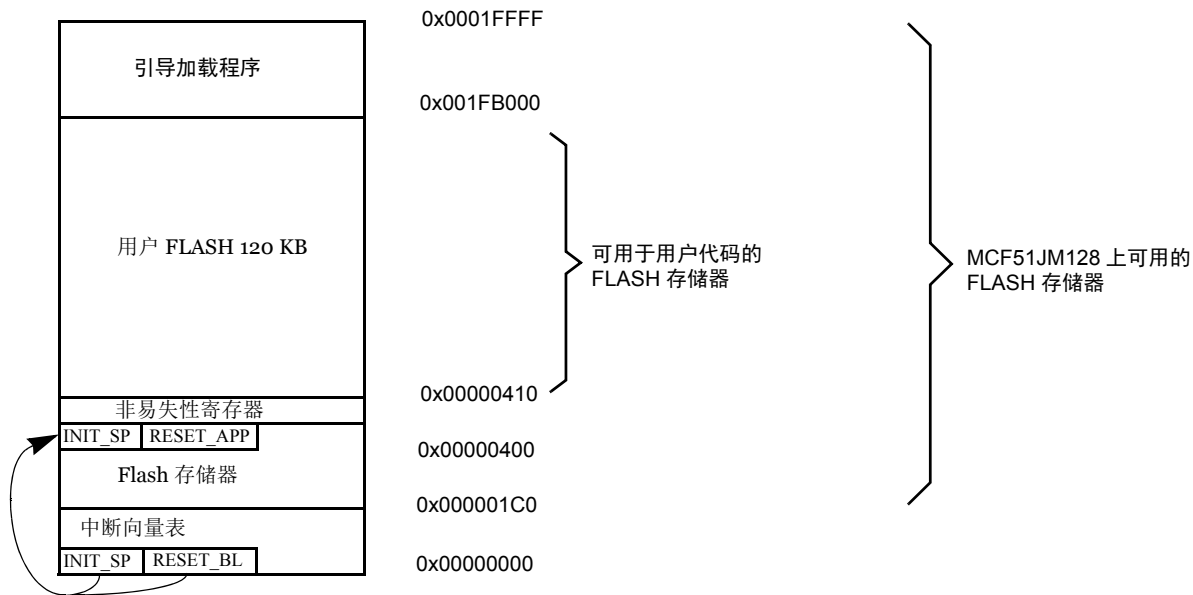


图 18. MCF51JM128 内存分配简单示例

6.1.1 内存分配

引导加载程序代码占用 FLASH 存储器的顶端（最高内存地址空间）。这种布置只减少顶部的内存空间，而且必须修改用户应用程序 LCF 文件的结尾；请参见[占用的内存](#)。

6.1.2 FLASH 保护

该版本的 MCU 从存储器的起始（对于 2 KB 扇区，起始地址为 0x0）支持 Flash 保护技术。

Flash 保护未在版本 A 协议中实施，因为该版本将地址 0x0 的原始向量表用于放置用户向量表。

6.1.3 IDENT 命令示例

ColdFire (V1) 引导加载程序的内存分配示例如下：

- \$84 - 版本 4，读命令已实施（位 7）
- \$rC16 - 系统器件识别寄存器 (SDIDR) 内容（\$C16 适用于 JM 系列，r（四个高位））为反映当前硅等级的芯片修订编号
- \$02 - 可重编程存储区的数目
- \$00000 - 可重编程内存区 1 的起始地址
- \$003FF - 可重编程内存区 1 的结束地址 + 1
- \$00410 - 可重编程内存区 2 的起始地址
- \$1FAFF - 可重编程内存区 2 的结束地址 + 2
- \$00000 - 重定位中断向量表的地址（值 0 表示未分配）

- \$00000 - MCU 中断向量表的起始地址 (值 0 表示未分配)
- \$00400 - MCU 擦除块的长度
- \$00080 - MCU 写块的长度
- ‘MCF51xxxx/USB’ - 以零结尾的识别字符串。将显示在 PC 屏幕上的信息

下图说明了 ColdFire V1 MCU 的中断向量表重定位:

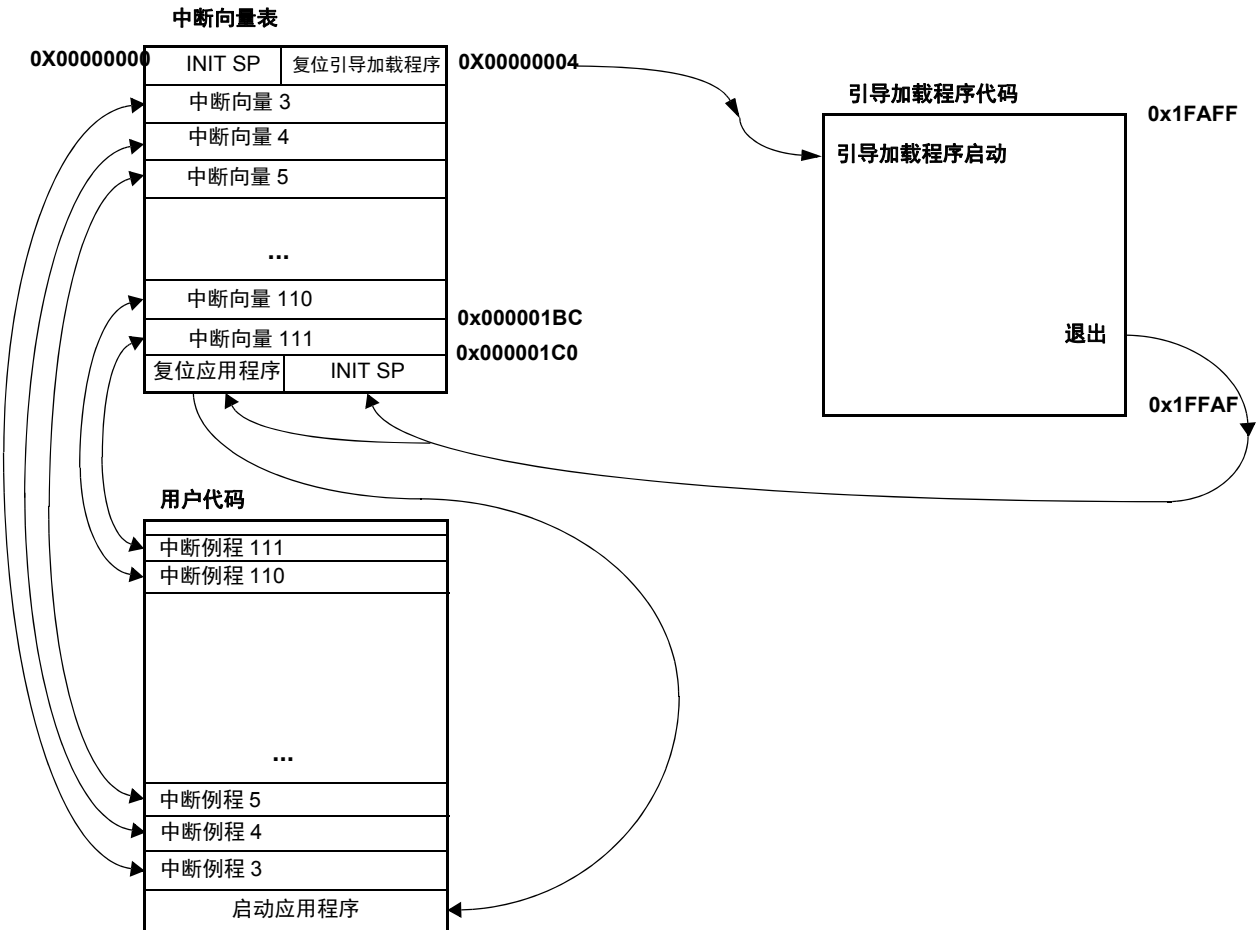


图 19. 中断向量表重定位说明 (ColdFire V1 - A)

6.1.4 软件复位

如果引导加载程序必须退出并运行用户代码, 则需要有意执行非法操作 (ColdFire 非法操作码停止 #0)。这会导致非法操作复位且 MCU 会重新启动。在引导加载程序启动过程中, 会检测“系统复位状态 (SRS)”寄存器。如果未检测到上电复位, 那么启动的会是用户代码, 而非引导加载程序代码。这就允许透明操作所有其他复位, 并且检测 SRS 寄存器和执行关联的跳跃指令只会引起较短的额外延迟。

6.1.5 ColdFire 系统限制

本节总结将引导加载程序与用户应用程序结合使用时应考虑的各种限制情况。

6.1.5.1 占用的内存

最重要的是要尽可能使用最小的代码。典型的 ColdFire V1 实施为 1 KB (SCI 版本) 和 8 KB (对于 JM 版本为 USB 版本)。对于 USB 版本, 源代码的最大部分被 USB 驱动程序占用 (5 KB)。

引导加载程序限制 Flash 存储器顶端, 因此, 必须使用修改后的链接器命令文件 (LCF) 用户文件。如果 LCF 文件未正确设置, 那么引导加载程序将显示警告且引导加载程序会被擦除。修改实例如下代码块中所示:

```
# Sample Linker Command File for CodeWarrior for ColdFire MCF51JM128
# Memory ranges
MEMORY {
    vectors      (RX)  : ORIGIN = 0x00000000, LENGTH = 0x00000200
    application  (RX)  : ORIGIN = 0x00000410, LENGTH = 0x0001ABEF //example of memory allocation
    buffer       (RWX) : ORIGIN = 0x00800000, LENGTH = 0x00000100
    userram      (RWX) : ORIGIN = 0x00800100, LENGTH = 0x00003F00
}
```

6.1.5.2 复位传输说明

原始向量 1(INIT SP) 和向量 2(RESET) 会被重新写到引导加载程序的复位值和栈指针初始化值。用户应用程序的开始值被写入地址 0x1C0 中, 而栈指针的初始化值被写入地址 0x1C4 中。这两个值在每次引导加载循环时经过重新编程被设置为当前应用程序值。

6.2 版本 B (受保护版)

本节介绍针对于 Cold Fire V1 实施版本 B 的功能。内存分配为 MCU 特定, 因此, 所有变量的含义都会在本节中进行说明。

图 20 介绍引导加载程序经过预编程的 ColdFire V1 器件的典型内存分配。例如, MCF51JM128 器件的内存映射包括:

- 128 KB FLASH 存储器 (\$00000000-\$0001FFFF)
- 16 KB 随机访问存储器 (RAM) (\$00800000-\$00803FFF)
- 16 字节易失性寄存器 (\$00000400-\$0000040F)

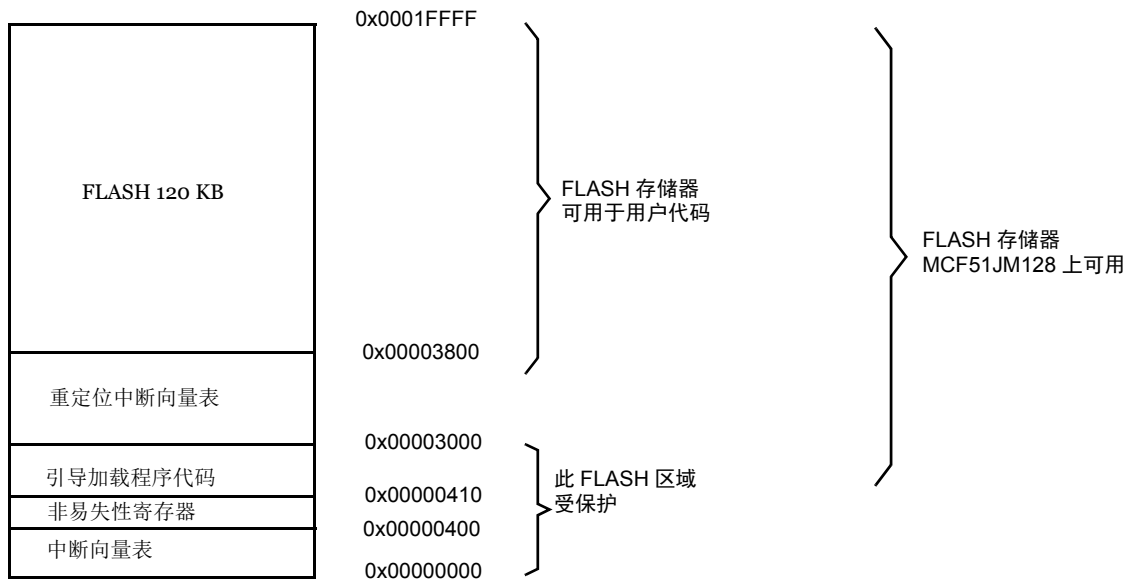


图 20. MCF51JM128 版本 B 内存分配简单实例

6.2.1 内存分配

引导加载程序代码占用 FLASH 存储器的底端, 范围为原始中断向量表以上的 0x0410 至 0x3000。这种布置可以移动开始部分的内存空间, 因此, 必须修改 LCF 文件 (详情请参见 MCU 特定数据手册)。

6.2.2 FLASH 保护

通过设置 FLASH 保护寄存器 (FPROT), 该地址以下的所有地址空间都会受到保护, 无法有意或无意擦除 / 重新写入。在引导加载程序和 FLASH 保护寄存器均被写入存储器中后, 引导加载程序代码会受到用户代码的保护以免发生无意修改。

6.2.3 内存分配实例

例如, ColdFire (V1) 引导加载程序的内存分配如下:

- \$84 - 版本 4, 读命令已实施 (位 7)。
- \$rC16 - 系统器件识别寄存器 (SDIDR) 内容 (\$C16 适用于 JM 系列, r (高四位)) 为反映当前硅等级的芯片修订编号。
- \$01 - 可重编程内存区的数目
- \$03800 - 可重编程内存区的起始地址。
- \$1FFFF - 可重编程内存区的结束地址 + 1。

- \$03000 - 重定位中断向量表的地址。
- \$001BC - MCU 中断向量表的起始地址。
- \$00400 - MCU 擦除块的长度。
- \$00080 - MCU 写块的长度。
- ‘MCF51JM128/USB’ - 以零结尾的识别字符串。将显示在 PC 屏幕上的信息。

6.2.4 限制

本节总结将引导加载程序与用户应用程序结合使用时应考虑的各种限制情况。

6.2.4.1 占用的内存

该版本的引导加载程序限制了 Flash 存储器的开端。因此, 用户必须修改链接器命令文件 (LCF) 并将用户 Flash 起点的边界移至地址 0x3800。以下代码实例所示为适合于用户应用程序的 LCF 文件。

```
# Sample Linker Command File for CodeWarrior for the ColdFire MCF51JM128
# Memory ranges

MEMORY {
application (RX) : ORIGIN = 0x00003800, LENGTH = 0x0001C7FF //memory allocation
userram (RWX) : ORIGIN = 0x00800000, LENGTH = 0x00003FFF
}
```

6.2.4.2 JMP 指令延迟

接下来的限制会增加中断延迟, 因为使用了两次跳转指令。完整情形如下图所示:

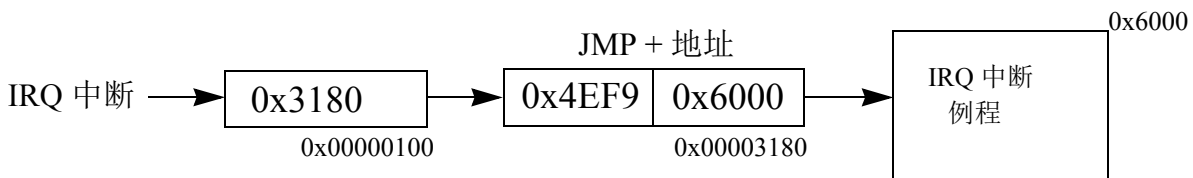


图 21. 使用 JMP 指令进行的向量重定向

下图说明的是中断向量表重定位 (ColdFire V1 - B):

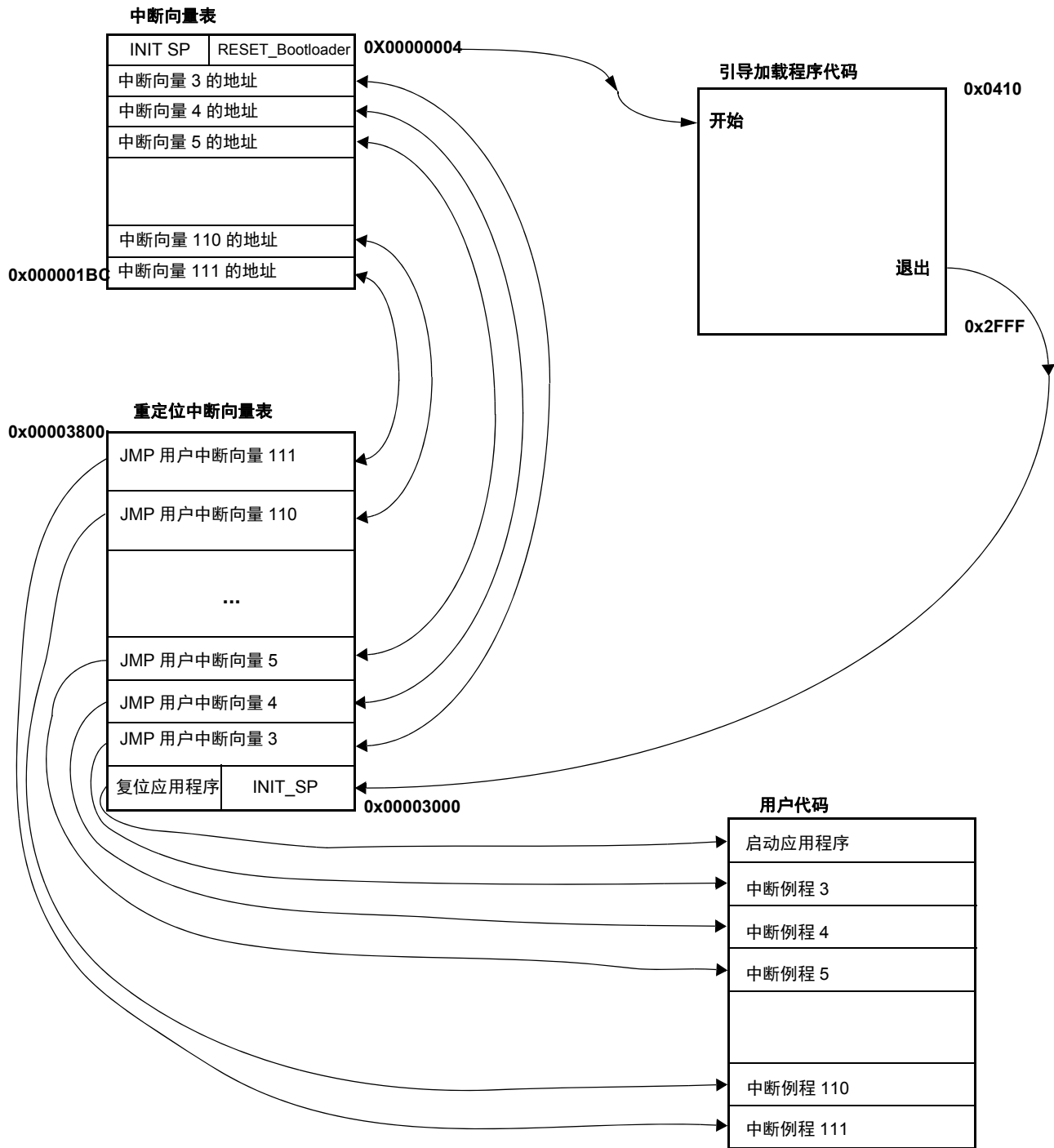


图 22. 中断向量表重定位说明 (ColdFire V1 - B)

7 FC 协议, 版本 5, Kinetis

本节介绍特定于引导加载程序协议版本 5 的功能。创建该版本是为了更好地兼容全新的 Kinetis 系列 MCU。适用于 ColdFire MCU 版本 B (受保护版) 的协议 4 是 Kinetis 协议版本 5 的基础。Kinetis MCU 的引导加载程序还具有 CRC 控制特性。内存分配高度取决于 MCU, 因此, 所有变量的含义在以下小节中会详细介绍。

图 23 介绍引导加载程序经过预编程的 Kinetis K60 器件的典型内存分配。例如, PK60N512 器件的内存映射包括:

- 495 KB FLASH 存储器 (\$00004000 - \$0007FFFF)
- 128 KB 随机访问存储器 RAM) (\$001FFE0000-\$002001FFFF)
- 16 字节易失性寄存器 (\$00000400-\$0000040F)
- 444 字节用户定义向量 (\$00000000-\$000001B8)

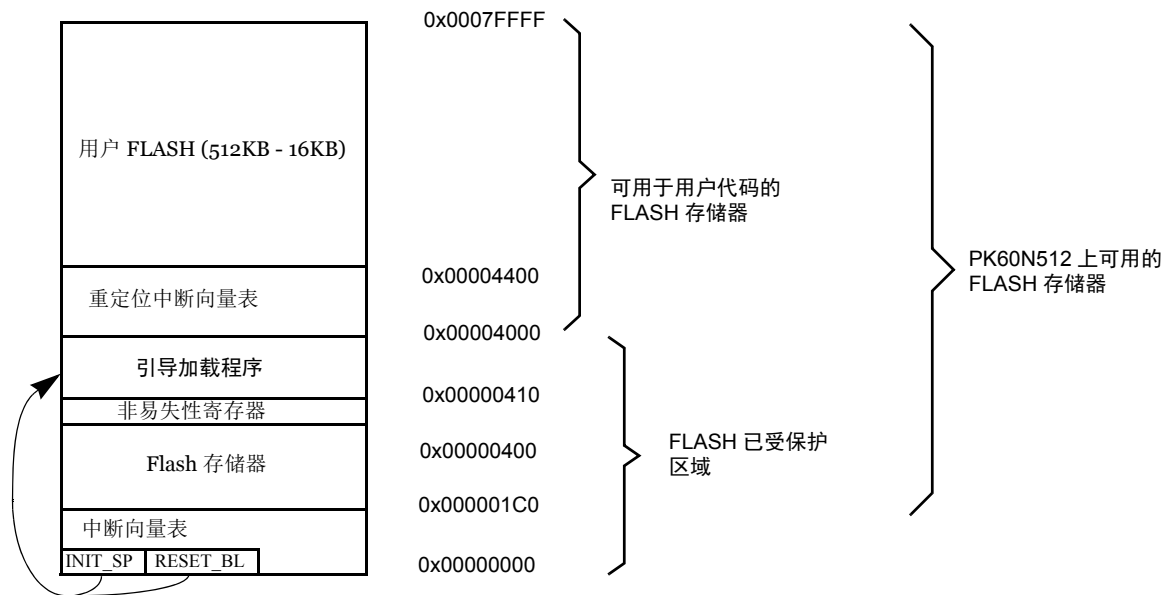


图 23. PK60N512 内存分配简单实例

7.1 内存分配

引导加载程序代码占用 FLASH 存储器的第一个区域 (最低内存地址空间)。该布置在可用内存空间的开端处移动, 而且有必要在用户应用程序链接器文件 (在 IAR 中为 ICF 文件, 在 CodeWarrior 中为 LCF 文件) 中进行该地址的修改。ICF 和 LCF 链接器文件修改示例如下:

Kinetis K60

示例: 修改 IAR6.4 中的 ICF 文件

```
// default linker file
define symbol __ICFEDIT_region_ROM_start__ = 0x00000000;
define symbol __code_start__ = 0x00000410;

// modified linker file for Kinetis K60 with 512KB flash memory
define symbol __ICFEDIT_region_ROM_start__ = 0x00004000;
define symbol __code_start__ = 0x000004400;
```

示例: 修改 CodeWarrior 10.2 中的 LCF 文件

```
# Default linker command file.
MEMORY {
m_interrupts (RX) : ORIGIN = 0x00000000, LENGTH = 0x000001E0
m_text (RX) : ORIGIN = 0x00000800, LENGTH = 0x00040000-0x00000800
m_data (RW) : ORIGIN = 0x1FFF8000, LENGTH = 0x00010000
m_cfmprotrom (RX) : ORIGIN = 0x00000400, LENGTH = 0x00000010
}
# Modified linker command file.
MEMORY {
m_interrupts (RX) : ORIGIN = 0x00000000, LENGTH = 0x000001E0
m_text (RX) : ORIGIN = 0x00004400, LENGTH = 0x00040000-0x00004400
m_data (RW) : ORIGIN = 0x1FFF8000, LENGTH = 0x00010000
m_cfmprotrom (RX) : ORIGIN = 0x00000400, LENGTH = 0x00000010
}
```

Kinetis KL25

示例: 修改 IAR6.4 中的 ICF 文件

```
// default linker file
define symbol __ICFEDIT_region_ROM_start__ = 0x00000000;
define symbol __code_start__ = 0x00000410;

// modified linker file for Kinetis K60 with 512KB flash memory
define symbol __ICFEDIT_region_ROM_start__ = 0x00001000;
define symbol __code_start__ = 0x0000010C0;
```

示例: 修改 CodeWarrior 10.2 中的 LCF 文件

```
# Default linker command file.
MEMORY {
m_interrupts (RX) : ORIGIN = 0x00000000, LENGTH = 0x000001E0
m_text (RX) : ORIGIN = 0x00000800, LENGTH = 0x00010000-0x00000800
m_data (RW) : ORIGIN = 0x1FFF8000, LENGTH = 0x00010000
m_cfmprotrom (RX) : ORIGIN = 0x00000400, LENGTH = 0x00000010
}
# Modified linker command file.
MEMORY {
m_interrupts (RX) : ORIGIN = 0x00000000, LENGTH = 0x000001E0
m_text (RX) : ORIGIN = 0x000010C0, LENGTH = 0x00010000-0x000010C0
m_data (RW) : ORIGIN = 0x1FFF8000, LENGTH = 0x00010000
m_cfmprotrom (RX) : ORIGIN = 0x00000400, LENGTH = 0x00000010
}
```

7.2 中断向量表重定向

由于 FLASH 块保护技术还保护中断向量表使其免受覆盖, 因此必须采用某种方法将这些向量重定位至不同位置。为此, 需使用引导加载程序用户表。

Flash 存储器开始的边界会移至 Flash 存储器第一个无保护区域的地址 (对于 512 KB Flash 存储器的 Kinetis K60 为 0x00004000) 处, 因为在此部分存储器下方放置的是受保护的引导加载程序。

7.3 FLASH 保护

Kinetis MCU 使用四个可实现 32 个可保护区域的 8 位寄存器为 Flash 保护提供支持。这四个寄存器中的每个位都保护可编程 Flash 存储器的 1/32 区域。例如, 对于具有 512 KB Flash 存储器的 Kinetis K60, 最小受保护区为 16 KB。为了实现引导加载程序的目的, 对于 512 KB 的 Kinetis K60 来说, 应使用地址 \$00000000-\$00003FFF 之间的第一个 Flash 内存区的保护区域。

下图所示为 Kinetis MCU 中的 Flash 存储器保护系统:

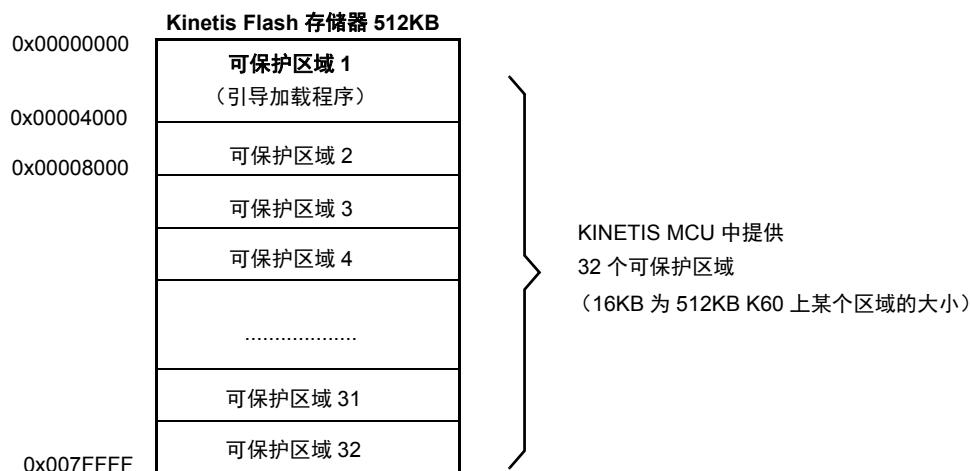


图 24. Kinetis MCU 中的 Flash 存储器保护系统

7.4 IDENT 命令示例

Kinetis K60 引导加载程序的内存分配示例如下:

- \$C8 - 版本 5, 读命令已实施 (位 8), CRC 已启用 (位 7)
- \$r14A - 系统器件识别寄存器 (SDID) 内容 (\$14A 适用于 K60 系列), r (13-16 位) 为反映当前硅等级的芯片修订编号
- \$01 - 可重编程内存区的数目
- \$0004400 - 可重编程内存区的起始地址
- \$007FFFF - 可重编程内存区的结束地址

- \$0000000 - 原始向量表的地址 (1KB)
- \$0004000 - 新向量表的地址 (1KB)
- \$00400 - MCU 擦除块的长度
- \$0080 - MCU 写块的长度

7.5 软件复位

如果引导加载程序必须退出并运行用户代码, 则可通过使用寄存器 AIRCR (应用中断和复位控制寄存器) 中的系统复位序列位有意执行 MCU 复位操作。在引导加载程序启动过程中, 会检测“系统复位状态 (SRS)”寄存器。如果未检测到上电复位, 那么启动的会是用户代码, 而非引导加载程序代码。这就允许透明操作所有其他复位, 并且检测 SRS 寄存器只会引起短时额外延迟。

7.6 Kinetis 系列限制

本节总结将引导加载程序与用户应用程序结合使用时应考虑的各种限制情况。

7.6.1 占用的内存

该版本的引导加载程序限制了 Flash 存储器的开端。因此, 必须存在适合于目标应用的已修改的命令链接器文件 (ICF), 并且必须将用户 Flash 存储器开端的边界移至受保护区域下面的地址 (例如, 对 K60 来说, 应移至地址 \$4400)。

下图说明了 Kinetis K60 MCU 的中断向量表重定位:

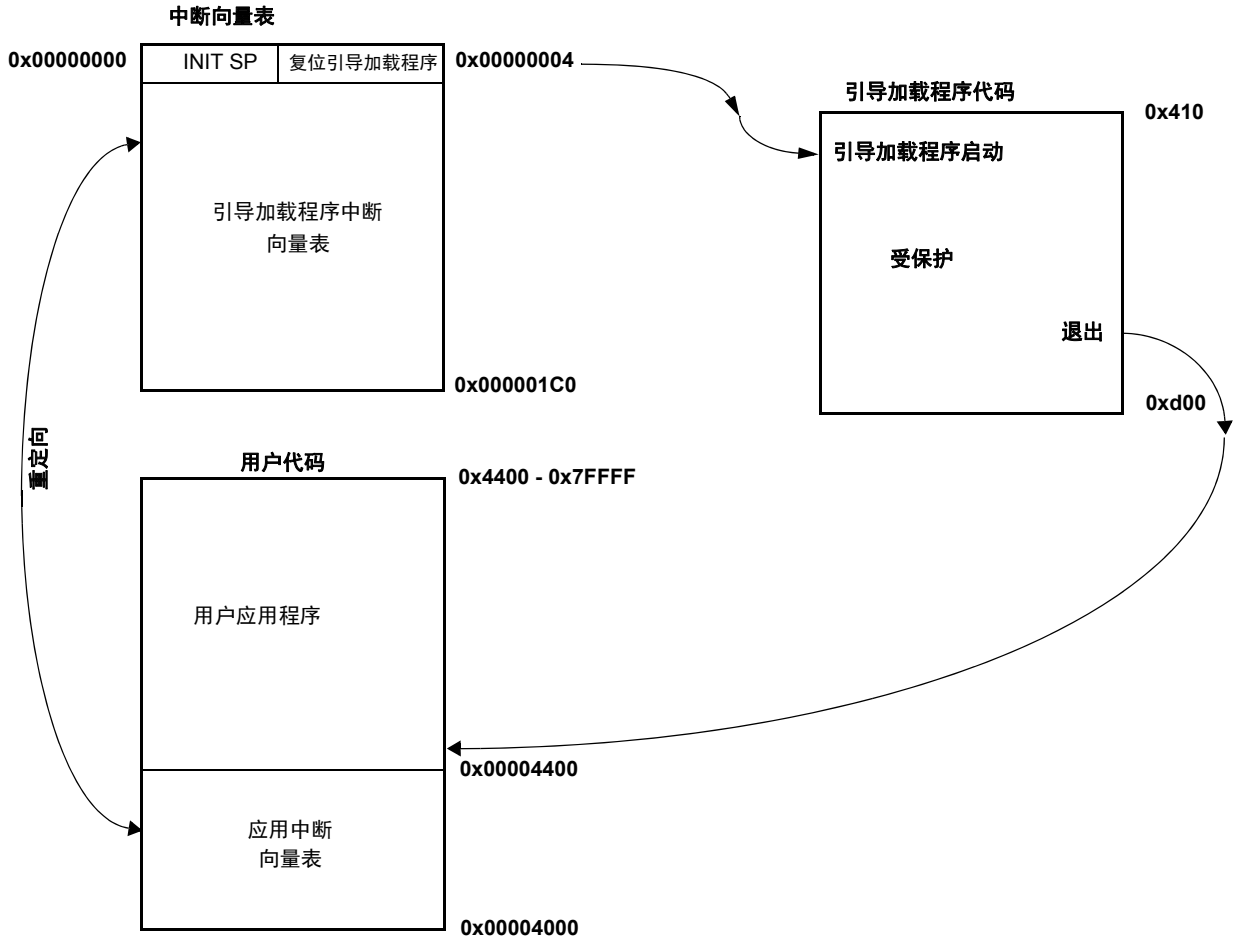


图 25. Kinetis 中断向量表重定位说明 (版本 5)

7.7 正确设置配置文件

引导加载程序配置文件 *bootloader_cfg.h* 可通过使用适合于许多 Kinetis 评估板的预定义配置文件为用户提供新的特性，并且还允许用户创建任何特定配置。

Configuration file *bootloader_cfg.h* (例如, 使用适合于 TOWER K60 板的预定义配置)

```
#ifndef KINETIS_K
#include "AN2295_TWR_K60_cfg.h"
#endif
```

对于实现正确的引导加载程序功能, 必须定义以下宏 (配置文件 *AN2295_TWR_K60_cfg.h*):

```
/* USER SETTINGS OF KINETIS MCU */
```

FC 协议, 版本 5, Kinetis

```
/** Kinetis ARM Cortex-M4 model */
//K10_50MHz K11_50MHz K12_50MHz K10_72MHz K10_100MHz K10_120MHz
//K20_50MHz K21_50MHz K22_50MHz K20_72MHz K20_100MHz K20_120MHz
//K30_72MHz K30_100MHz
//K40_72MHz K40_100MHz
//K50_72MHz K51_72MHz K50_100MHz
//K60_100MHz K60_120MHz
//K70_120MHz

/** Kinetis ARM Cortex-M0+ model */
//KLO_48MHz
//KL1_48MHz
//KL2_48MHz KL25_48MHz

#define KINETIS_MODEL K60_100MHz

/* in the case of using USB VIRTUAL SERIAL LINK you must activate No break TRIM CHECKBOX in
the master AN2295 PC Application */
/* the break impulse is replaced by using only 9 bits zero impulse */
// BREAK IMPULSE      |START| |0| |0| |0| |0| |0| |0| |0| |0| |0| |STOP|
// ZERO IMPULSE       |START| |0| |0| |0| |0| |0| |0| |0| |0| |0| |STOP|

#define BOOTLOADER_SHORT_TRIM 1

Kinetis flash memory can be defined in different sizes and supported sizes are 32, 64, 128,
256, 512 and 1024 KBytes.
#define KINETIS_FLASH FLASH_512K

Following define determines if the bootloader code will be protected or not (protection is
recommended). First section of the flash will be protected (protected_size = flash_size / 32).
protection enabled - 1 , protection disabled - 0
#define BOOTLOADER_FLASH_PROTECTION 1

Flash write access allows change mode of access to flash memory. Each model of MCU can support
different write access. Supported write access macros are defined as follows:

FLASH_WRITE_ACCESS_LONG - 32 Bytes
FLASH_WRITE_ACCESS_PHRASES - 64 Bytes
FLASH_WRITE_ACCESS_DOUBLE_PHRASES - 128 Bytes

#define FLASH_WRITE_ACCESS FLASH_WRITE_ACCESS_PHRASES

Address of base pointer to actual used UART module
#define BOOT_UART_MODULE UART2_BASE_PTR

Range of UART baudrates is between (9600 - 115200 Baud)
#define BOOT_UART_BAUD_RATE 115200

Address of peripheral base pointer for GPIO port (number of GPIO port shared with UART module)
#define BOOT_UART_GPIO_PORT PORTE_BASE_PTR

Setting of multiplexer for UART alternative of the pin
#define BOOT_PIN_UART_ALTERNATIVE 3

Setting of multiplexer for GPIO alternative of the pin
#define BOOT_PIN_GPIO_ALTERNATIVE 1
```

```

Number of UART & GPIO pin for receiver (Rx)
#define BOOT_UART_GPIO_PIN_RX 17

Number of UART & GPIO pin for transmitter (Tx)
#define BOOT_UART_GPIO_PIN_TX 16

/*****
/* Actual used PIN reset setting */
#define BOOT_PIN_ENABLE_PORT_BASE PORTC_BASE_PTR

#define BOOT_PIN_ENABLE_GPIO_BASE PTC_BASE_PTR

#define BOOT_PIN_ENABLE_NUM 9

以下宏允许使用引导加载程序的自身特性:
Read command feature allows to check the flash memory.
#define BOOTLOADER_ENABLE_READ_CMD 1

Watchdog timer can be enabled or disabled.
#define BOOTLOADER_INT_WATCHDOG 0

Verification of memory without CRC functions
#define BOOTLOADER_ENABLE_VERIFY 1

Verification of memory with CRC functions
#define BOOTLOADER_CRC_ENABLE 1

Autotrimming function allows to calibrate internal oscillator of MCU. If these feature is not
enabled user must define your own clock initialization or trimming of internal oscillator.
#define BOOTLOADER_AUTO_TRIMMING 1

This feature allows using external pin for the bootloader starting
#define BOOTLOADER_PIN_ENABLE 0

/*****
/** CALIBRATION OF BOOTLOADER TRIM SETTINGS */
Address of flex timer base pointer
#define BOOT_CALIBRATION_TIMER FTM0_BASE_PTR

Address of GPIO PORT base pointer
#define BOOT_CALIBRATION_GPIO_PORT PTE_BASE_PTR

```

7.8 快速指南：如何为 AN2295 引导加载程序准备 Kinetis 用户应用

修改用户应用程序以准备运行 AN2295 引导加载程序时，必须注意以下三种限制：

1. 链接器文件：必须将用户应用程序移至引导加载程序代码上面。以下规则可告知用户应用程序需移至的目标位置：
 - Flash (2048 B Flash 保护块) 大于 / 等于 64 KB 的 MCU: 在此情况下，用户应用程序的开端必须从第二个保护块加向量表大小开始。向量表基本上都应当放置在第二个保护块的开端。
 - Falsh 小于 64KB 的 MCU: 在此情况下，用户应用程序应当从 0x800 开始，且中断向量和应用程序应处于中断表上面。

- 要更为详细地修改链接器文件，请参见[内存分配](#)。
- 2. Flash 配置寄存器：Kinetis 中的 Flash 配置（保护、安全和其他）置入地址 0x400 处，因为该地址位于引导加载程序代码的区域，应当从用户应用程序中删除这些寄存器的定义。
- 3. VTOR 寄存器：某些用户应用程序将 VTOR（向量表偏移寄存器）寄存器设置为以默认值 (0x0000) 启动，因此，可以将此寄存器配置删除或更新为指向当前使用的向量表（一般情况下为用户应用程序的第一个字节）。

7.9 使用适合于 MQX 应用程序的 Kinetis 引导加载程序

Kinetis 引导加载程序可用于对 MQX 应用程序进行编程。该主题介绍在 MQX 应用程序中必须完成的准备用于 AN2295 引导加载程序的操作。

以下步骤与[快速指南：如何为 AN2295 引导加载程序准备 Kinetis 用户应用](#)节所述的用户应用程序相似，MQX 应用程序无需修改 VTOR 寄存器。

必须更新 MQX 项目中的以下两个内容：

1. 链接器文件：对于 MQX 的链接器文件，其情况与裸机用户应用程序相似。必须将 MQX 应用程序移至 AN2295 引导加载程序代码上面。MQX 链接器文件使用已有的标准链接器定义。例如，针对 K60N512 的 IAR6.4 工具的链接器文件的更新行如下所示：
 - `define symbol __ICFEDIT_intvec_start__ = 0x00004000;`
 - `define symbol __ICFEDIT_region_ROM_start__ = 0x00004000;`
 - `define exported symbol __INTERNAL_FLASH_BASE = 0x00004000;`
 - `define exported symbol __VECTOR_TABLE_ROM_START = 0x00004000;`
2. Flash 配置寄存器：在 MQX 应用程序中设置 Flash 配置寄存器比在一般裸机应用中简单。只需定义一个 MQX 宏即可：
 - `#define BSPCFG_ENABLE_CFMPROTECT 0`

附注

可能会存在禁用具有某些 BSP 的 CFMPROTECT，但这种情况可以单独解决。

8 MCU 从机软件

本节详细介绍五种典型 M68HC(S)08、Cold Fire V1 和 Kineti 引导加载程序的实施。所有代码均以汇编语言编写。下表列出了几种所选目标芯片及其不同的特性：

表 2. 目标实施比较

| MCU 系列 | FLASH 存储器使用量 (单位: 字节) | 时钟源 | ROM 例程与 硬件 | 校准与 否执行 | SCI | FLASH 擦 除页大小 (单位: 字 节) | FLASH 编 程页大小 (单位: 字 节) |
|--------------------------------------------------|--------------------------|------------------------------------------------|------------------|------------|--------------------|---------------------------------|---------------------------------|
| MC68HC908AP AP8/AP16/ AP32/AP64 | 592 | 32768 Hz XTAL 或外部时钟。 | 是, 但 版本不 同 | 否 | 硬件 | 512 | 64 |
| MC68HC908AB/AS/AZ AB32/AS32/AZ32 AS60/AZ60 | 640 | 4.9152MHz XTAL | 否 | 否 | 硬件 | 128 | 64 |
| MC68HC908EY EY16 | 384 | ICG | 是 | 是 | 硬件 | 64 | 32 |
| MC68HC908GP GP32 | 512 | 32768 Hz XTAL 或外部时钟。 | 否 | 否 | 硬件 | 128 | 64 |
| MC68HC908GR GR4/GR8/GR16 GR8A/GR16A | 320 | 32768 Hz XTAL 或外部时钟; 8MHz XTAL (A 系列) | 是 | 否 | 硬件 | 64 | 32 |
| MC68HC908GT GT8/GT16 | 384 | ICG | 是 | 是 | 硬件 | 64 | 32 |
| MC68HC908GZ GZ8/GZ16 | 512 | 8 MHz XTAL | 是 | 否 | 硬件 | 64 | 32 |
| MC68HC908GZ GZ60 | 512 | 8 MHz XTAL | 否 | 否 | 硬件 | 128 | 64 |
| MC68HC908JK/JL JK1/JL1/ JK3/JL3 | 395 | XTAL、RC 振荡器 或外部源 | 是 | 是 | 软件, 可 采用单线 式 | 64 | 32 |
| MC68HC908JK/JL JK8/JL8 | 384 | 4.9152MHz XTAL | 是, 但 版本不 同 | 否 | 硬件 | 64 | 32 |
| MC68HC908JW JW32 | 1968 | 4MHz 或 6MHz XTAL 或谐振器 | 是 | N/A | USB2.0 | 512 | 64 |
| MC68HC908LB LB8 | 384 | ICG | 是 | 是 | 软件, 可 采用单线 式 | 64 | 32 |
| MC68HC908LJ LJ12/ LJ/LK24 | 324 | 32768 Hz XTAL 或外部时钟。 | 是, 但 版本不 同 | 否 | 硬件 | 128 | 64 |
| MC68HC908KX KX2/KX8 | 384 | ICG | 是 | 是 | 硬件 | 64 | 32 |
| MC68HC908MR MR8 | 461 | PLL, 具有 XTAL (4 MHz) | 否 | 否 | 硬件 | 64 | 32 |
| MC68HC908MR MR16/MR32 | 461 | PLL, 具有 XTAL (4 MHz) | 否 | 否 | 硬件 | 128 | 64 |

开发人员的串行引导加载程序, Rev. 13

表 2. 目标实施比较 (续)

| MCU 系列 | FLASH 存储器使用量 (单位: 字节) | 时钟源 | ROM 例程与 使用 | 校准与 执行 | SCI | FLASH 擦 除页大小 (单位: 字 节) | FLASH 编 程页大小 (单位: 字 节) |
|-------------------------------------------------------------------------------------|--------------------------|---------------|---------------|------------------------|--------------------|---------------------------------|---------------------------------|
| MC68HC908QB QB4/QB8 | 362/302 | QB/QC ICG | 是 | 是 / 否 | 硬件 | 64 | 32 |
| MC68HC908QC QC8/QC16 | 387/323 | QB/QC ICG | 是 | 是 / 否 | 硬件 | 64 | 32 |
| MC68HC908QT/QY QT1/QT4/ QY1/QY4 | 320 | 更简单的 ICG | 是 | 是 | 软件, 可 采用单线 式 | 64 | 32 |
| MC68HC908SR SR12 | 512 | 32768 Hz XTAL | 否 | 否 | 硬件 | 128 | 64 |
| MC9S08AW HCS08AW32/48/64 | 576 | HCS08 ICG | 否 | 是 | 硬件 | 512 | 64 |
| HCS08AC8 HCS08AC16 HCS08AC32 HCS08AC48 HCS08AC60 | 432 | HCS08 ICG | 否 | 是 | 硬件 | 512 | 64 |
| HCS08AC128 | 694 | HCS08 ICG | 否 | 是 | 硬件 | 512 | 128 |
| MC9S08GB/GT HCS08GB/GT32 HCS08GB/GT60 | 576 | HCS08 ICG | 否 | 是 | 硬件 | 512 | 64 |
| HCS08QE4 HCS08QE8 HCS08QE16 HCS08QE32 | 432 | HCS08 ICG | 否 | 否 | 硬件 | 512 | 64 |
| MC9S08QG HCS08QG4/8 | 576 | HCS08 ICG | 否 | 否 (硬件) 是 (软件) | 硬件 软件 | 512 | 64 |
| MC9S08Rx HCS08RD/RG/RE8 HCS08RD/RG/RE16 HCS08RD/RG/RE32 HCS08RD/RG/RE60 | 335 | 16MHz XTAL | 否 | 否 | 硬件 | 512 | 64 |
| HCS08JM32 HCS08JM60 | 6000 | 12MHz 外部时钟 | 否 | 否 | USB 2.0 | 512 | 64 |
| MCF51JM64 MCF51JM128 | 1108 | S08 MCGV3 | 否 | 否 | 硬件 | 1024 | 128 |
| MCF51QE32 MCF51QE64 MCF51QE128 | 1104 | S08 ICSV3 | 否 | 否 | 硬件 | 1024 | 128 |

表 2. 目标实施比较 (续)

| MCU 系列 | FLASH 存储器使用量 (单位: 字节) | 时钟源 | ROM 例程与 使用 | 校准与 执行 | SCI | FLASH 擦 除页大小 (单位: 字 节) | FLASH 编 程页大小 (单位: 字 节) |
|----------------------------------------------------|----------------------------------------|------------|---------------|-----------|---------|---------------------------------|---------------------------------|
| MCF51CN64 MCF51CN128 | 1132 | MCG | 否 | 否 | 硬件 | 1024 | 128 |
| MCF51AC128 MCF51AC256 | 1116 | MCG | 否 | 否 | 硬件 | 1024 | 128 |
| MCF51AG96 MCF51AG128 | 1120 | ICS | 否 | 否 | 硬件 | 1024 | 128 |
| MCF51EM128 MCF51EM256 | 1284 | ICS | 否 | 否 | 硬件 | 1024 | 128 |
| MCF51JM64 MCF51JM128 | 1116 | MCG | 否 | 否 | 硬件 | 1024 | 128 |
| MCF51JM64 MCF51JM128 | 8000 | 12MHz 外部时钟 | 否 | 否 | USB 2.0 | 1024 | 128 |
| K10N1M0 K10N512 K10N256 K10N128 K10N64 | 32768 16384 8192 4096 2048 | MCG | 否 | 否 | 硬件 | 2048 | 128 |
| K20N512 K20N256 K20N128 K20N64 K20N32 | 16384 8192 4096 2048 1024 | MCG | 否 | 否 | 硬件 | 2048 | 128 |
| K30N512 K30N256 | 16384 8192 | MCG | 否 | 否 | 硬件 | 2048 | 128 |
| K40N512 K40N256 K40N128 | 16384 8192 4096 | MCG | 否 | 否 | 硬件 | 2048 | 128 |
| K50N512 K50N256 | 16384 8192 | MCG | 否 | 否 | 硬件 | 2048 | 128 |
| K60N1024 K60N512 | 32768 16384 | MCG | 否 | 否 | 硬件 | 2048 | 128 |
| K70N1M | 32768 | MCG | 否 | 否 | 硬件 | 2048 | 128 |
| K22N1M0 | 32768 | MCG | 否 | 否 | 硬件 | 2048 | 128 |
| KL05Z32 | 2048 | MCG | 否 | 否 | 硬件 | 1024 | 128 |
| KL25Z128 | 4096 | MCG | 否 | 否 | 硬件 | 1024 | 128 |
| KM34Z128 | 4096 | MCG | 否 | 否 | 硬件 | 1024 | 128 |

8.1 MC68HC908KX

M68HC908KX 系列具有一个内部时钟发生器 (ICG) 模块。在没有晶振的情况下，该模块对引导加载程序的实施非常有效。

FLASH 片上编程例程可简化引导加载程序并提高存储器的使用效率。MCU 与 PC 间的通信采用标准串行通道 (SCI)。

以下流程图介绍引导加载算法的基本原理：

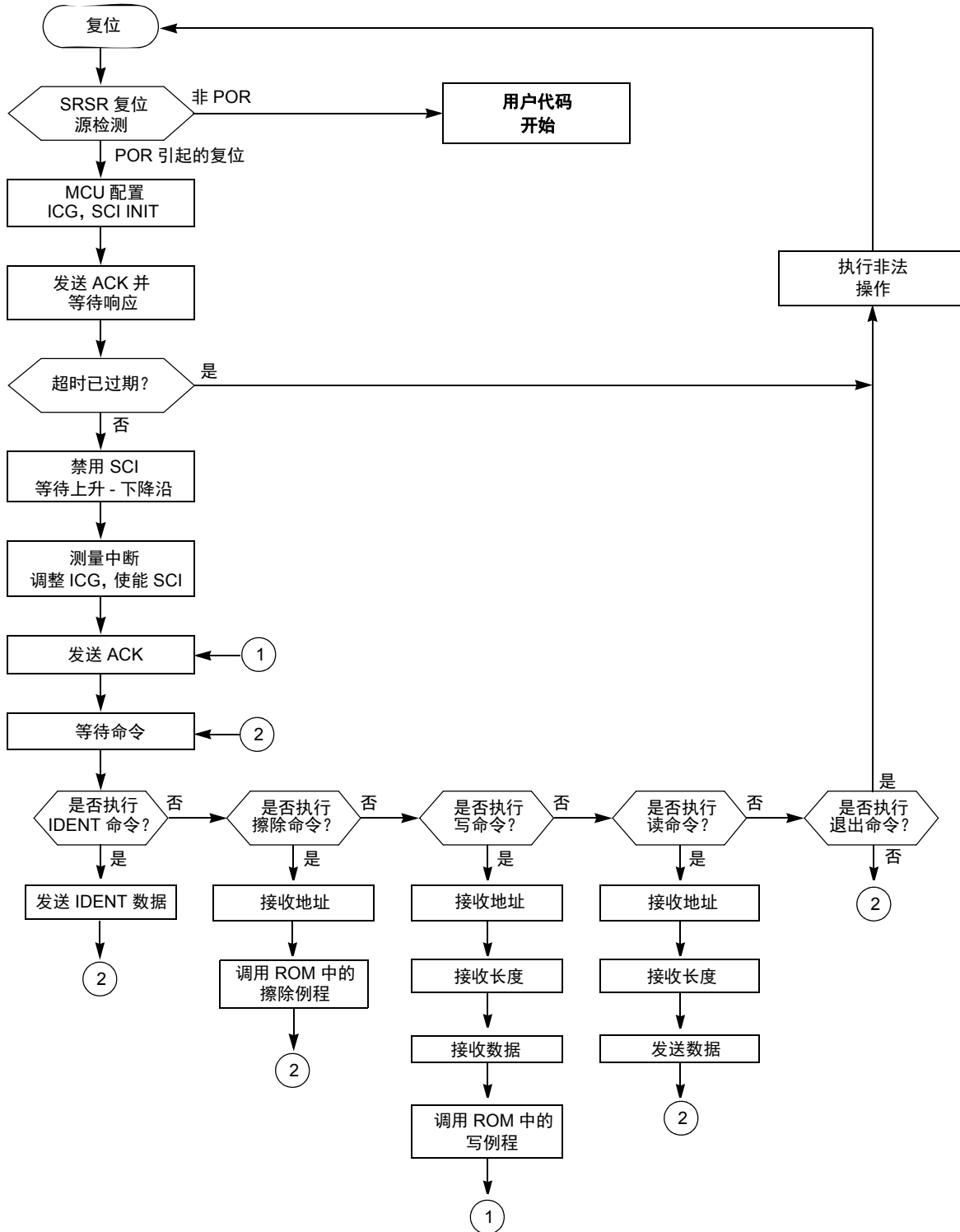


图 26. MC68HC908KX 引导加载程序流程图

8.1.1 内部时钟发生器 (ICG) — 初始化

ICG 的初始化很简单，因为复位后，ICG 处于激活状态并且时钟监控已被禁用。唯一需要做的就是修改 ICG 倍频寄存器。然后，ICG 控制寄存器的 ICGS 标志（第 2 位）即表示 ICG 在频率改变后是否稳定。

```
ICGMRINIT      EQU      $20

                MOV      #ICGMRINIT,ICGMR      ; set 9.8304MHz BUS clock
LOOP:          BRCLR   2,ICGCR,LOOP           ; wait until ICG stable
```

8.1.2 内部时钟发生器 — 调整

即便调整例程位于 ROM 中，但一个小小的缺陷就会导致该代码不可用；因此采用源代码并将其嵌入到引导加载程序代码中。

尽管 AN1831/D 提供了根据测得的 CPU 速度计算调整因子的程序，但是代码本身忽略了最后将周期数翻倍。

```
* FOLLOWING LOOP IS EXECUTED UNTIL THE END OF THE BREAK SIGNAL. THE BREAK
* SIGNAL LASTS 10 BIT TIMES. IF COMMUNICATING AT f OP /256 BPS, THEN 10 BIT
* TIMES IS 2560 CYCLES. EACH TIME THROUGH THE LOOP IS 10 CYCLES, SO WE
* EXPECT TO EXECUTE THE LOOP 256 TIMES IF THE KX8 IS IN SYNC SERIALY WITH
* THE HOST. IF WE STAY IN THE LOOP FOR > 256 LOOP CYCLES, THEN THE KX8
* MUST BE RUNNING FASTER THAN EXPECTED, AND NEEDS TO BE SLOWED DOWN. IF WE
* STAY IN THE LOOP FOR < 256 LOOP CYCLES THEN THE KX8 MUST BE RUNNING SLOWER
* THAN EXPECTED AND NEEDS TO BE SPEEDED UP. THE AMOUNT THAT WE CHANGE THE
* CPU SPEED IS EQUAL TO THE NUMBER OF LOOP CYCLES OVER OR UNDER 256. SO IF
* WE GO THROUGH THE LOOP 240 TIMES, THEN WE ARE RUNNING
* (256-240)/256 = 6.25% FAST. EACH INCREMENTAL CHANGE WE MAKE TO THE TRIM REGISTER
* (ICGTR) WILL MAKE A 0.195% CHANGE TO THE INTERNAL CLOCK. THAT IS, INCREMENTING
* THE REGISTER BY ONE OVER THE DEFAULT VALUE OF $80 STORED THERE WILL
* DECREASE THE INTERNAL CLOCK BY 0.195%, AND VICE VERSA.
* NOW EACH EXECUTION OF THE LOOP OVER OR UNDER WHAT IS EXPECTED (256 TIMES)
* REPRESENTS AN ERROR OF 1/256 = .391% ERROR. SO WE'LL NEED TO DOUBLE THE
* NUMBER OF LOOP CYCLES AND USE THIS NUMBER TO CORRECT THE TRIM REGISTER.
* OUR PRECISION FOR TRIMMING IS THEREFORE 0.391%.
```

实际代码增加了一条 ASLA 指令，用于在实际写入 ICG 调整寄存器之前使调整因子加倍。

```
ICGTRIM:
    CLRX
    CLRH

MONPTB4:
    BRSET 4,PTB,MONPTB4 ;WAIT FOR BREAK SIGNAL TO START
CHKPTB4:
    BRSET 4,PTB,BRKDONE ; (5) GET OUT OF LOOP IF BREAK IS OVER
    AIX #1 ; (2) INCREMENT THE COUNTER
    BRA CHKPTB4 ; (3) GO BACK AND CHECK SIGNAL AGAIN
BRKDONE:
    PSHH
    PULA ;PUT HIGH BYTE IN ACC AND WORK WITH A:X
    TSTA ;IF MSB OF LOOP CYCLES = 0, THEN BREAK TAKES TOO
    TXA ;FEW CYCLES THAN EXPECTED, SO TRIM BY SPEEDING
    BEQ SLOW ;UP f OP .
```

```

FAST:  CMP    #$40          ;SEE IF BREAK IS WITHIN TOLERANCE
      BGE    OOR           ;DON'T TRIM IF OUT OF RANGE
      ASLA          ;multiply by two to get right range
      ADD    #$80          ;BREAK LONGER THAN EXPECTED, SO SLOW DOWN f OP
      BRA    ICGDONE
SLOW:  CMP    #$C0          ;SEE IF BREAK IS WITHIN TOLERANCE
      BLT    OOR           ;DON'T TRIM IF OUT OF RANGE
      ASLA          ;multiply by two to get right range
      SUB    #$80
ICGDONE:
      STA    ICGTR
OOR:
      RTS

```

有关调整程序的完整说明在 AN1831/D 中给出。请参见[参考文献](#)。

8.2 MC68HC908JK/JL

MC68HC908JK/JL 系列是 M68HC08 系列中最便宜的 MCU，不过没有硬件 SCI。因此，必须实施软件 SCI。这样，对于串行通信选用哪些引脚就没有任何限制（规定均通过代码实现，因此 IRQ 引脚还可用作输入串行线）。

MC68HC908JK/JL 系列具有 RC 版本（使用的是 RC 振荡器，而不是晶振）。任何速度变化均可通过校准引导加载程序进行补偿。如果期望的时钟频率超出校准系统所覆盖的范围，则必须修改代码。

MC68HC908JK/JL 系列具有片上 FLASH 编程例程。使用 FLASH 编程可以节省内存。

主程序流程图（图 27）与先前情况非常相似。下图所示为 MC68HC908JK/JL 引导加载程序流程图：

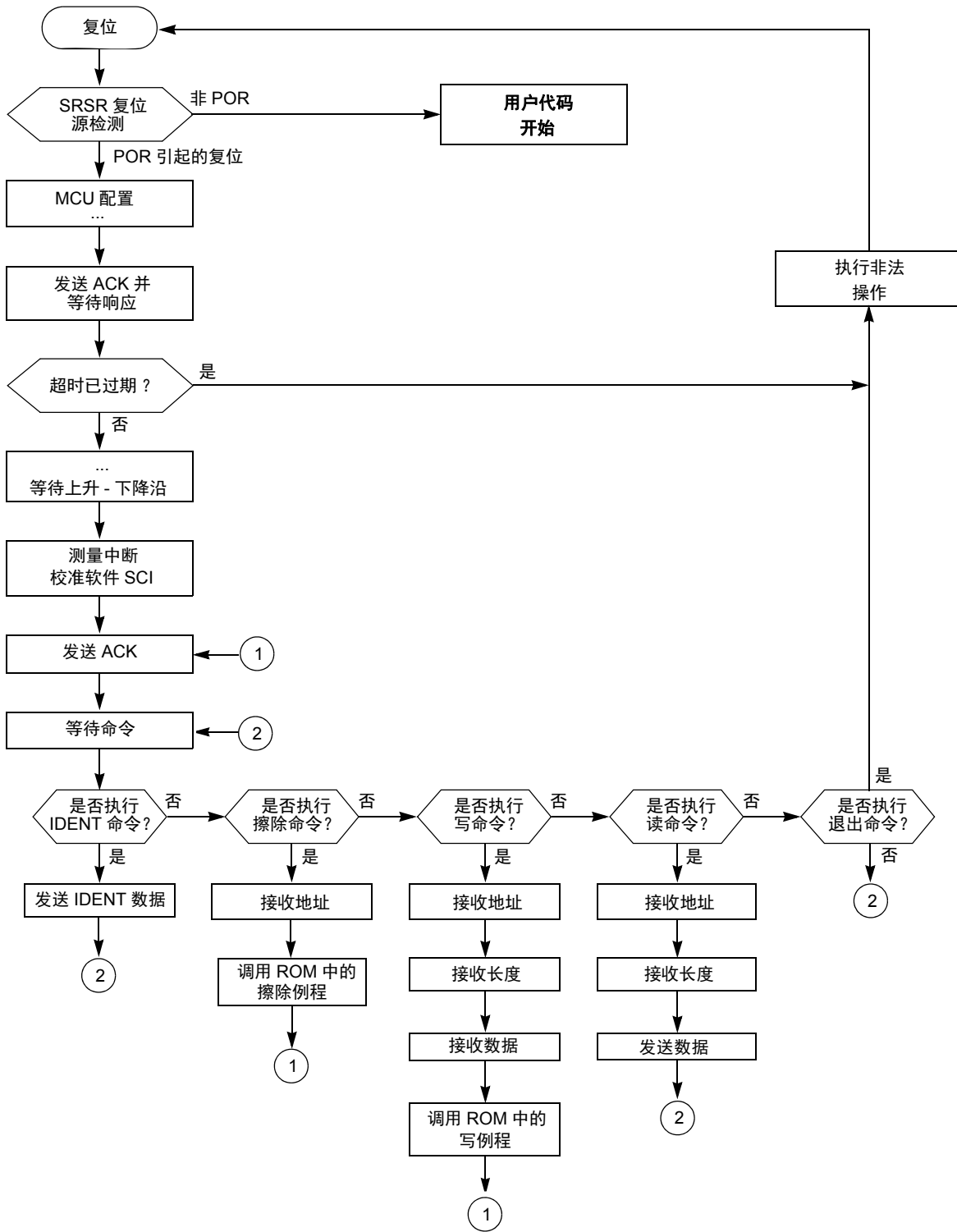


图 27. MC68HC908JK/JL 引导加载程序

8.2.1 软件 SCI 的字符传送例程

本节详细说明软件 SCI 的传送和接收子程序。这两个程序都基于 16 位定时器，在后台循环中轮询输出比较事件。

以下所示为软件 SCI 的字符传送程序流程图：

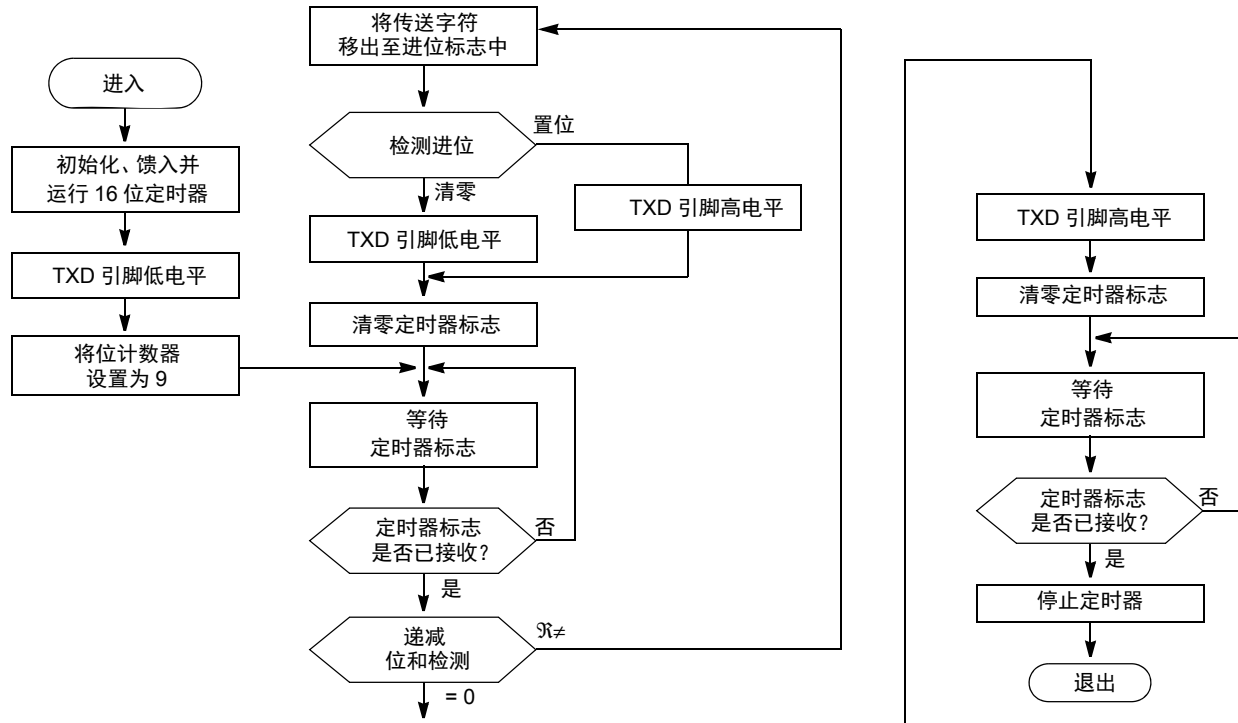


图 28. 软件 SCI 的字符传送例程

这两个程序的源代码如图 29 所示。除使用几个计数器外，还使用一个 16 位 ONEBIT 变量。它包含在 16 位定时器时钟周期中处于当前通信速度时的 1 位实际长度。该变量在校准阶段被初始化（从机频率校准）。

以下所示为软件 SCI 的字符传送程序的源代码：

```

;*****
SCITX:
    PSHH
    PSHX

    BCLR    7,TSC           ; and clear TOF
    LDHX   ONEBIT
    STHX   TMOD
    BSET   4,TSC           ; clear timer
    BCLR   5,TSC           ; run timer

    TXDCLR

    MOV    #9,BITS        ; number of bits + 1
    BRA   SCITX1          ; jump to loop

SCITX2:
    LSRA                   ; shift out lowest bit
    BCC   DATALOW

    TXDSET
    SKIP2                   ; skip next two bytes
DATALOW:
    TXDCLR

    BCLR   7,TSC           ; and clear TOF
SCITX1: BRCLR 7,TSC,SCITX1 ; wait for TOF

    DBNZ  BITS,SCITX2     ; and loop for next bit

SCISTOP:
    TXDSET

    BCLR   7,TSC           ; and clear TOF
SCITX3: BRCLR 7,TSC,SCITX3 ; wait for TOF
EPILOG:
    BSET   5,TSC           ; stop timer

    PULX
    PULH
    RTS

```

图 29. 软件 SCI 的字符传送程序源代码

8.2.2 软件 SCI 的字符接收程序

软件 SCI 的字符接收程序与软件 SCI 的字符传送程序相似。当轮询到 16 位输出比较事件时，会扫描接收引脚的值。未针对停止位校验、成帧检查、噪声检测等制定规定，这主要是因为存在内存限制。下图所示为软件 SCI 接收程序流程图，图 31 中提供的是其源代码。

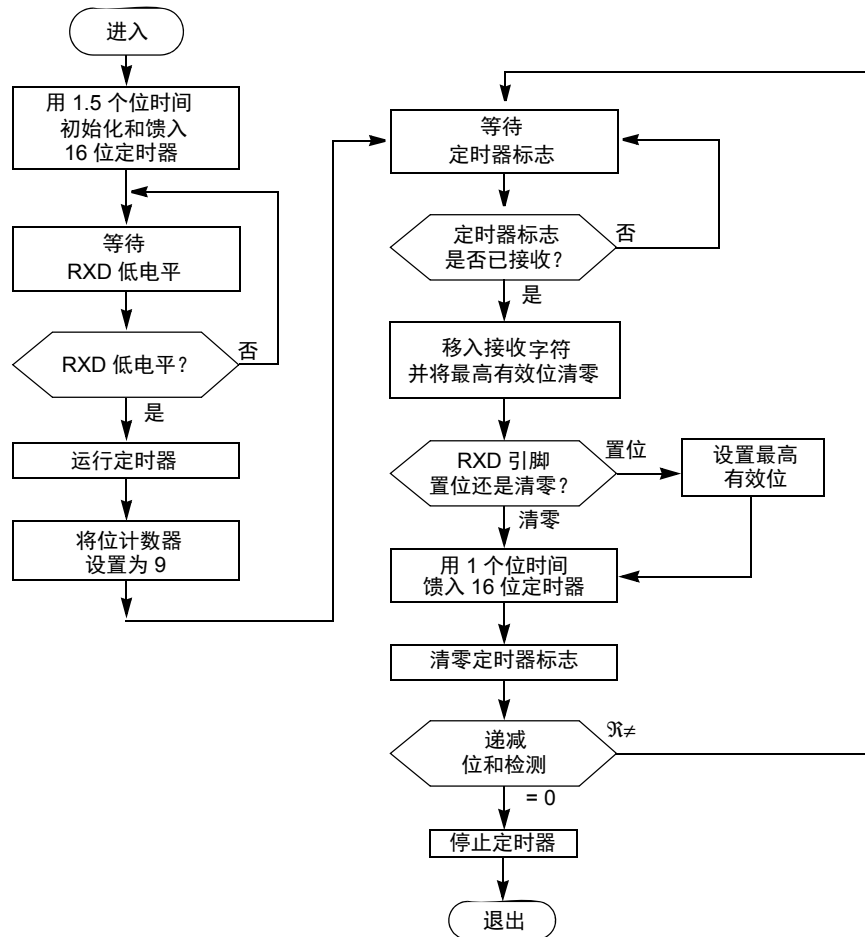


图 30. 软件 SCI 的字符接收程序

```

;*****
SCIRX:
    BRRXDLO SCIRX          ; loop until RXD high (idle)

SCIRXNOEDGE:
    PSHH
    PSHX
    BCLR    7,TSC          ; and clear TOF

    LDX    ONEBIT
    LDA    ONEBIT+1
    LSRX
    RORA
    STX    TMODH
    STA    TMODL

    BSET    4,TSC          ; clear timer

SCIRX1:
    BRRXDHI SCIRX1        ; loop until RXD low (wait for start bit)

    BCLR    5,TSC          ; run timer
    MOV     #9,BITS        ; number of bits + 1

SCIRX2: BRCLR    7,TSC,SCIRX2 ; wait for TOF

    LSRA          ; shift data right (highest bit cleared)
    BRRXDLO RXDLOW ; skip if RXD low
    ORA    #$80    ; set highest bit if RXD high

RXDLOW: LDHX    ONEBIT
    STHX    TMOD

    BCLR    7,TSC          ; and clear TOF
    DBNZ   BITS,SCIRX2    ; and loop for next bit

    BRA    EPILOG

```

图 31. 软件 SCI 的字符接收程序的源代码

8.2.3 宏

这两个代码列表中定义了几个宏。它们改善了程序的可读性以及内存消耗（图 32）。以下所示为软件 SCI 宏的源代码：


```

SKIP1          MACRO
                  DC.B    $21          ; BRANCH NEVER (saves memory)
                  ENDM

SKIP2          MACRO
                  DC.B    $65          ; CPHX (saves memory)
                  ENDM

BRRXDLO        MACRO

    IFNE    RXDISIRQ
    IFNE    SCIRXINV
        BIH    \1          ; branch if RXD low
    ELSE
        BIL    \1          ; branch if RXD low
    ENDIF
    ELSE    ; RXD uses normal I/O pin
    IFNE    SCIRXINV
        BRSET  RXDPIN,RXDPORT,\1    ; branch if RXD low
    ELSE
        BRCLR  RXDPIN,RXDPORT,\1    ; branch if RXD low
    ENDIF
    ENDIF

    ENDM

BRRXDHI        MACRO

    IFNE    RXDISIRQ
    IFNE    SCIRXINV
        BIL    \1          ; branch if RXD hi
    ELSE
        BIH    \1          ; branch if RXD hi
    ENDIF
    ELSE    ; RXD uses normal I/O pin
    IFNE    SCIRXINV
        BRCLR  RXDPIN,RXDPORT,\1    ; branch if RXD hi
    ELSE
        BRSET  RXDPIN,RXDPORT,\1    ; branch if RXD hi
    ENDIF
    ENDIF

    ENDM

TXDCLR         MACRO

    IFNE    SCITXINV
        BSET  TXDPIN,TXDPORT ; clr bit
    ELSE
        BCLR  TXDPIN,TXDPORT ; clr bit
    ENDIF

    ENDM

TXDSET         MACRO

    IFNE    SCITXINV
        BCLR  TXDPIN,TXDPORT ; set bit
    ELSE
        BSET  TXDPIN,TXDPORT ; set bit
    ENDIF

    ENDM

```

图 32. 软件 SCI 宏的源代码

8.3 MC68HC908GP

MC68HC908GP 系列 MCU 没有可用的片上 FLASH 编程例程。因此，如本节所述，所有 FLASH 编程均由引导加载程序完成。

MC68HC908GP 系列 MCU 主要是与低成本的 32.768 kHz 晶振结合使用。由于晶振的频率是已知的，因此无需进行校准，从而可节省 MCU 内存。因此，此类 MCU 使用的是[已知 MCU 通信速度方法](#)。

下图所示为 MC68HC908GP 引导加载程序的流程图：

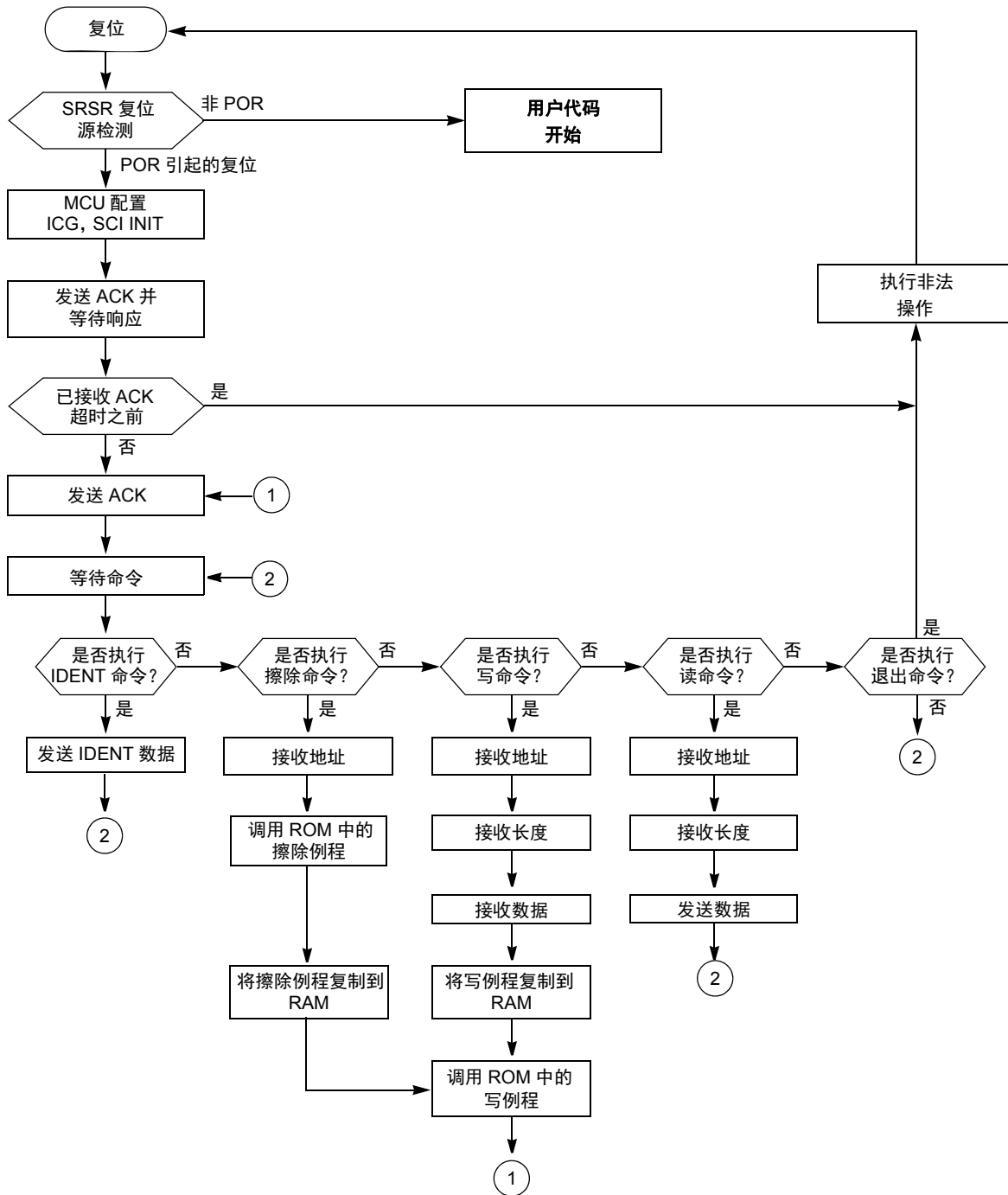


图 33. MC68HC908GP 引导加载程序流程图

8.3.1 FLASH 编程例程

主要代码与前面省略的校准阶段的实施类似。通过引导加载程序实现的 FLASH 编程如图 34 所示。三个主要子程序的定义如下：

- CPY_PRG — 将所选的例程复制到 RAM 中

MCU 从机软件

- ERASE_ALG — 整个 FLASH 擦除例程
- WR_ALG — 整个 WRITE 擦除例程

鉴于流程很简单，故未提供流程图。事件的执行顺序基本上符合 FLASH 擦除 / 编程规范。

```
*****
;*****
CPY_PRG:
    TSX                ;
    STHX   STACK      ; copy stack for later re-call

    LDHX   SOURCE     ; LOAD WRITE ALGORITHM TO RAM
    TXS
    LDHX   #PRG
CPY_PRG_L1:
    PULA
    STA    X
    AIX    #1
    DBNZ   STAT,CPY_PRG_L1

    LDHX   STACK
    TXS                ; restore stack
    RTS
;*****
ERASE_ALG:

    LDA    #%00000010
    STA    FLCR        ; ERASE bit on
    LDA    FLBPR       ; dummy read FLBPR

    LDHX   ADRS        ; write anything
    STA    X            ; to desired range
    D_US   #T10US      ; wait 10us

    LDA    #%00001010
    STA    FLCR        ; set HVEN, keep ERASE
    D_MS   #T1MS       ; wait 1ms

    LDA    #%00001000
    STA    FLCR        ; keep HVEN, ERASE off
    D_US   #T5US       ; wait 5us

    CLRA
    STA    FLCR        ; HVEN off
    D_US   #T1US       ; wait 1us

    JMP    SUCC        ; finish with ACK
ERASE_ALG_END:
;*****
WR_ALG:
    LDA    #%00000001
    STA    FLCR        ; PGM bit on
    LDA    FLBPR       ; dummy read FLBPR

    LDHX   ADRS        ; prepare addresses
    STA    X            ; and write to desired range
    D_US   #T10US      ; wait 10us
```

```

        LDA    #%00001001
        STA    FLCR           ; set HVEN, keep PGM
        D_US   #T5US         ; wait 5us

        LDHX   #DAT           ; prepare addresses
        TXS
        LDHX   ADRS
        MOV    LEN,POM

WR_ALG_L1:
        PULA
        STA    X
        AIX    #1
        D_US   #T30US        ; wait 30us
        DBNZ   POM,WR_ALG_L1 ; copy desired block of data

        LDA    #%00001000
        STA    FLCR           ; keep HVEN, PGM off
        D_US   #T5US         ; wait 5us

        CLRA
        STA    FLCR           ; HVEN off
        D_US   #T1US         ; wait 1us

        JMP    RETWR         ; finish with ACK (& restore STACK before)
WR_ALG_END:
END

```

图 34. FLASH 编程例程的源代码

为了增强程序的可读性，代码（图 35）中使用了两个用于定时的宏（D_US 和 D_MS）。

```

;*****
D_MS:   MACRO
        LDA    \1           ; [2] ||
\@L2:   CLRX           ; [1] ||
\@L1:   NOP           ; [1] |
        DBNZX  \@L1      ; [3] |   256*4 = 1024T
        DBNZA  \@L2      ; [3] ||  (1024+4)*(arg-1) + 2 T
        ENDM

D_US:   MACRO
        LDA    \1           ; [2]
\@L1:   NOP           ; [1]
        DBNZA  \@L1      ; [3] 4*(arg-1) + 2 T
        ENDM

```

图 35. FLASH 编程宏的源代码

8.4 MC68HC908GR

MC68HC908GR 系列 MCU 比 MC68HC908GP 系列 MCU 的内存空间小，它的 ROM 存储器中带有用户模式下可用的 FLASH 编程例程。

MC68HC908GP 和 MC68HC908GR 系列 MCU 主要是与低成本的 32.768 kHz 晶振结合使用。由于晶振的频率是已知的，因此无需进行校准，从而可节省 MCU 内存。因此，这些 MCU 使用的是已知 MCU 通信速度方法。

8.5 MC68HC908MR

MC68HC908MR 系列 MCU 是 M68HC08 系列中面向电机控制的成员。MC68HC908MR 系列 MCU 没有可用的片上 FLASH 编程例程。因此，所有 FLASH 编程均由引导加载程序完成。

MC68HC908MR 系列 MCU 中具有一个 PLL（锁相环）电路，可使晶振频率倍增。通常情况下，将 4MHz 的 XTAL 用作参考频率。此实施说明了如何将 PLL 电路初始化为 8 倍的晶振频率。因此，源 PLL 频率为 32MHz，总线频率为 8MHz。

由于晶振的频率是已知的，因此无需进行校准，从而可节省 MCU 内存。因此，这些 MCU 使用的是已知 MCU 通信速度方法。

8.6 MC68HC908EY

除内存映射和 ROM 例程地址不同外，MC68HC908GT 和 MC68HC908EY 系列 MCU 的代码与 MC68HC908KX 代码相似。存在的细小差别是 MC68HC908GT 系列不能将 CGMXCLK 时钟用作 SCI 模块的时钟源。因此，总线时钟是唯一可用的时钟源。

8.7 MC68HC908QT/QY

MC68HC908QT/QY 系列 MCU 是 M68HC08 系列中最小的成员。它们具有一个简单的 ICG 模块（以固定频率 $12.8\text{MHz} \pm 25\%$ 运行）。ROM 例程可用。

此外，还有几个备用 FLASH 区（主要在未使用的中断向量中）用于存放引导加载程序代码。

8.7.1 SCI 应用程序接口 (SCI API)

软件 SCI 通信可在 MC68HC908QT/QY、MC68HC908JK/JL 和 MC68HC908 上实施，以降低成本并让用户代码能够调用 SCI 传送和接收程序（有一定局限性）。引导加载程序代码现在实施的是 SCI API，它是以自定义的方法调用 SCI 传送和接收程序。

（QTQY 文件夹中的）sci.h 文件给出了详细信息、注意事项和局限性。该文件是唯一一个必须包含到用户 C 代码中的资源。该文件中还介绍调用规范以及所有的使用方法。对大多数应用来说，主要限制因素在于 SCI 接收例程是一个阻塞程序。也就是说，例程在接收到 SCI 字符之前不会返回。此外，还使用了 16 位的定时器寄存器。某些应用程序在使用该代码时不会出现问题。

8.7.2 单线通信

由于 MC68HC908QT 器件上的引脚数少，因此已开发出单线 SCI 版本使通信占用的引脚数最少。图 36 所示为单线 RS-232 接口的实例。MC68HC908JK/JL 和 MC68HC908LB 引导加载程序也使用软件 SCI，因此在它们的引导加载程序代码中加入了单线选项。

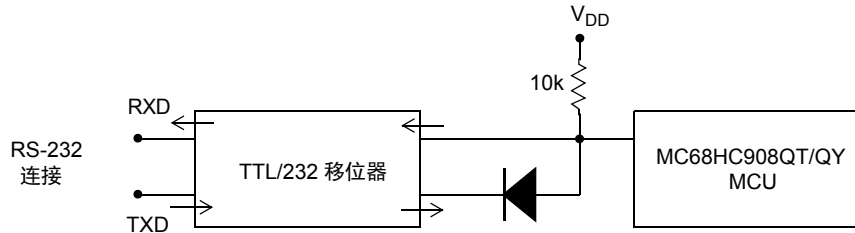


图 36. 单线实例原理图

使用单线通信时必须通知引导加载程序的主机方。这可以通过调用 `hc08sprg.exe` 软件完成。使用下列扩展调用协定：

```
hc08sprg.exe 1:S filename.s19
```

其中，1 指定通信使用的 COM 端口号，S 则表示单线。

原来的（旧）格式：`hc08sprg.exe 1 filename.s19`

现在默认为：`hc08sprg.exe 1:D filename.s19`

其中，D 表示双线模式。引导加载程序主机还能在实施以下调用时检测单线接口：

```
hc08sprg.exe 1:? filename.s19
```

仅当串行接口（主要为电平转换器）在引导加载流程开始前已上电且处于工作状态时才能进行检测。这种情况并不经常出现，因此，请务必通过在参数中加入“:S”或“:D”来指定引导加载模式。

8.8 MC68HC908LJ

MC68HC908LJ MCU 是 M68HC08 系列中用于驱动 LCD 显示器的成员。MC68HC908LJ MCU 具有可用的 ROM 片上 FLASH 编程例程。其调用协定与其他 M68HC08 略有不同（请参见 MC68HC908LJ 数据手册中的“监控 ROM”部分）。

MC68HC908LJ MCU 主要是与低成本的 32.768 kHz 晶振结合使用。由于晶振的频率是已知的，因此无需进行校准，从而可节省 MCU 内存。因此，这些 MCU 使用的是[已知 MCU 通信速度方法](#)。

8.9 MC68HC908AP

MC68HC908AP 器件是 M68HC08 系列中具有两个 SCI 的成员（编译时必须选择 SCI 通道）。MC68HC908AP MCU 具有可用的 ROM 片上 FLASH 编程例程。其调用协定与其他 M68HC08 略有不同（与 MC68HC908LJ 器件相同）。

内部振荡器较为简单，故其并不具备 RC 振荡器或 XTAL 振荡器的精度和稳定性。因此，在需要精确总线时钟的情况下，不适合使用内部振荡器，而且不应当将内部振荡器作为总线时钟源。

引导加载程序 and 用户应用程序的默认时钟源选用的是低成本的 32.768 kHz 晶振。由于晶振的频率是已知的，因此无需进行校准，从而可节省 MCU 内存。因此，这些 MCU 使用的是[已知 MCU 通信速度方法](#)。

8.10 MC68HC908AB/AS/AZ

MC68HC908AB/AS/AZ 器件是 M68HC08 系列中同样具有 EEPROM 存储器的成员。此代码也说明了如何使用 AUTO（自动清除 EEPGM）模式对这些 EEPROM 单元进行编程。

内存映射不是连续的，因此，需要使用第 3 版的 FC 协议（它允许内存映射中有“空洞”，即几个分开的内存区）。

8.11 MC9S08GB/GT

MC9S08GB/GT 是 HCS08 系列中的第一批成员。由于硬件特性和 FLASH 内存分配不同，因此需要另一版本的协议。该协议由最新的 PC 引导加载程序软件 hc08sprg.exe 自动检测并对用户可见。

MC9S08GB/GT MCU 具有 2 个 SCI（编译时必须选择 SCI 通道）。

这些 MCU 没有片上 FLASH 编程例程。因此，引导加载程序必须完成所有 FLASH 编程操作，该实施说明了此操作（全部来自 HCS08 系列参考手册卷 1（飞思卡尔半导体订单号为 HCS08RMv1/D；请参见参考文献））。

8.12 MC68HC908JW

HC908JW 系列具有内置的 USB2.0 全速模块。因此，通过真正的 USB 接口能与 PC 直接连接。如 [AN3153: Using the Full-Speed USB Module on the MCHC908JW32](#) 应用笔记所述，模拟串行 COM 口的设计很简单。这种方法也用于为 JW32 系列设计出一个完全兼容的引导加载程序（用 C 语言编写）。引导加载程序被编入 JW32 器件中后，可使用本地 USB 连接（Windows 中的模拟串行 COM 端口）随时对用户代码进行重编程。

[ZSTARRM: Wireless Sensing Triple Axis Reference design](#) 的第 5.5 节和第 6.1.2 节介绍安装和使用详情，JW32 USB 引导加载程序亦源于此。USB 所需的 PC 驱动程序也位于 AN2295SW 软件包的 JW32 文件夹中。而最新的在线版 PC 驱动程序在 ZSTAR 总结页面上也有提供（[RD3152MMA7260Q](#)）。

附注

尽管 JW32 的串行 COM 仿真已在 Linux 环境下检测成功，但 AN2295 主机引导加载程序的 hc08sprg 可执行文件的 Linux 端口尚未与 JW32 引导加载程序 USB 实施进行联合检测。

8.13 HCS08JM 和 MCF51JM

MCF51JM 系列具有内置的 USB2.0 全速模块。因此，通过真正的 USB 接口能与 PC 直接连接。如 [AN3492: USB 和使用 CMX USB 堆栈应用笔记](#) 所述，可轻松设计串行 COM 端口的仿真。这种方法也用于为 HCS08JM 和 MCF51JM 系列设计出一个完全兼容的引导加载程序（用 C 语言编写）。该 USB 驱动加载程序用于与 PC 通信器件类 (CDC) 通信。此类通信的基础在于在 PC 上创建虚拟串行端口。该功能允许在未修改源代码的情况下使用主机引导加载程序软件。有关更新信息，请参阅 [AN3492: USB 和使用 CMX USB 协议栈](#)。

9 PC 引导加载程序主机软件

本节详细介绍引导加载程序主机软件，采用 zip 格式的该软件可从飞思卡尔半导体的主页 freescale.com 下载。所有代码均用 C 语言编写，兼容 Linux 和 Win32 平台。

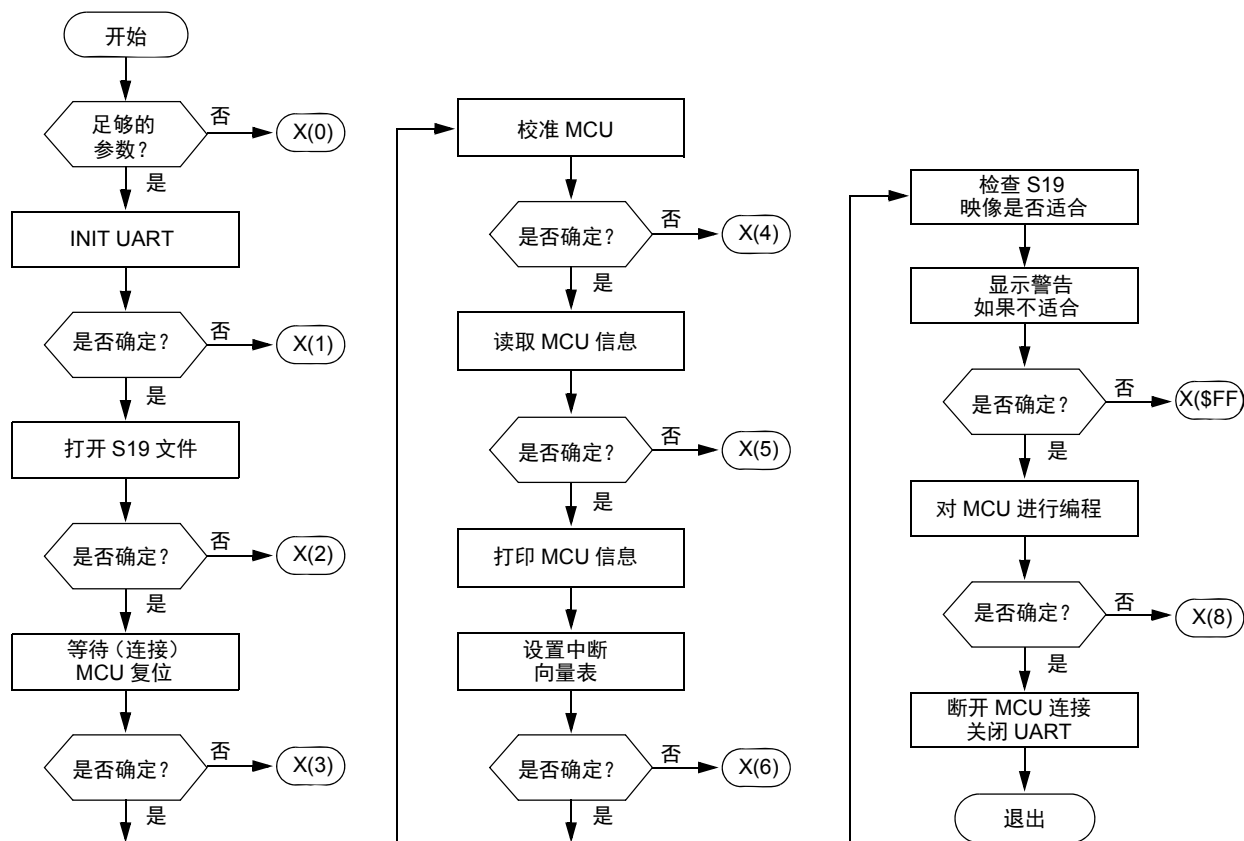
引导加载程序规范指出，为了最大限度的减少 MCU 内存消耗，智能功能在主机执行而非在 MCU 中执行。MCU 仅实施一些简单的功能。

本节更为详细地介绍主机引导加载程序中的部分代码。本文档的从机实施和协议部分介绍了对 M68HC(S)08 器件进行重编程所需的全部操作。着重强调了主机的具体特性。

主机控制软件的设计非常简单，只需要以下几个步骤：

- 打开串口
- 打开源 S19 文件
- 等待 MCU 复位
- 校准 MCU
- 读取 MCU 信息
- 重映射 MCU 中断向量
- 检查源 S19 数据是否能载入 MCU 的物理内存
- 对 MCU 进行擦除和编程
- 清理，退出程序

下图所示为引导加载程序主机的流程图：



注意：X(2) 表示通过退出代码 2 退出

图 37. 引导加载程序主机流程图

9.1 文件结构

PC 主机软件中使用的是以下文件结构设置：

- 8 位和 32 位 MCU 映像操作：
 - s19.c
- UART 处理：
 - serial.h
- seriallinux.c (serialw32.c)
 - 系统平台相关的文件：
- sysdep.h
 - sysdeplinux.h
 - sysdepw32.h
- 通用文件和主程序文件：
 - hc08sprg.h
 - main.c

- M68HC(S)08、ColdFire 和 Kinetis 的特定编程文件：
 - prog.c

9.2 8 位和 32 位 MCU 映像操作

为执行必要的代码操作，主机软件保持了存储器的二进制映像。此外，它还存储了有关是否要将实际字节写入 MCU 的信息。这可通过使用以下结构实现：

```
typedef struct {
    BYTE d[0x10000];    // data
    BYTE f[0x10000];    // valid flag 0=empty; 1=usercode; 2=systemcode
} BOARD_MEM;
```

其中，image 是一个实际变量，定义如下：

```
BOARD_MEM image;
```

读取完源 S19 文件后，此数组中会包含要写入 MCU 的实际数据，而无论其在 S19 文件中的最初顺序如何。s19.c 中定义的函数 `int read_s19(char *fn)` 实施 S19 文件的打开、读取以及从十六进制格式到此二进制数组的重定位操作。

9.2.1 中断向量表重定位

MCU 读出识别信息后，执行映像中的以下操作：

- 扫描代码，以确定 MCU 中断向量表地址与 0xFFFF (M68HC(S)08 MCU 的最后物理地址) 之间是否存在任何中断向量。
- 如果存在中断向量，则按照[中断向量表重定位](#)中所述完成这些向量的重定位。然后将中断向量表中的原始地址空间标为未用，这样就不能对其进行重新编程。

这些操作由 prog.c 文件中定义的函数 `int setup_vect_tbl(void)` 执行。

9.2.2 检查内存边界

在代码被真正写入 MCU 之前，执行的最后检查是确定 S19 文件中的代码是否位于正确的存储器位置（在识别表中 MCU 报告的内存边界之间）。

如果发现任何值超出该地址范围（可重编程内存区的起始地址与结束地址之间），则会生成警告。

此项检查也是由 prog.c 文件中定义的函数 `int check_image(void)` 完成。

9.3 UART 处理

在 seriallinux.c 或 serialw32.c（取决于所使用的平台）中，定义了以下 UART 处理函数：

```
int init_uart(char* nm);
int close_uart(void);
int send_break10(void);
int flush_uart(int out, int in);
int wb(const void* data, unsigned len);
int rb(void* dest, unsigned len);
```

`int init_uart(char* nm)` 和 `int close_uart(void)` 管理指定 UART 端口的打开（初始化）和关闭。

`int wb(const void* data, unsigned len)` 和 `int rb(void* dest, unsigned len)` 用于将数据块写入 UART 以及从 UART 中读出数据块。

另外两个函数 `int send_break10(void)` 和 `int flush_uart(int out, int in)` 是引导程序工作所必需的。第一个函数将 BREAK 字符发送到 UART，第二个函数清空 UART 两个方向（入/出）的缓冲区。

9.4 系统平台相关的文件

头文件 `sysdep.h` 包含 `sysdeplinux.h` 或 `sysdepw32.h`，具体取决于软件编译的平台。然后是使用的特定平台声明。

9.5 通用文件和主程序文件

头文件 `hc08sprg.h` 包含编译应用程序所需的其他通用声明。文件 `main.c` 包含主程序，它已在本节开始处给出（图 37）。

9.6 M68HC(S)08、ColdFire 和 Kinetis 的特定编程文件

PC 引导加载程序软件的最重要部分包含在文件 `prog.c` 中，它实施先前章节所述的 PC 引导加载程序软件的大部分信息。

许多例程均是在文件 `prog.c` 中实施：

```
int hook_reset(void)
int could_be_ack(unsigned b)
int calibrate_speed(void)
int read_mcu_info(void)
int setup_vect_tbl(void)
int check_image()
int read_blk(unsigned adr, int len, BYTE *dest)
int erase_blk(unsigned a)
int prg_blk(unsigned a, int len)
int prg_area(unsigned start, unsigned end)
int prg_mem(void)
int erase_mem(unsigned all)
int verify_mem(int byS19_range)
int prg_only_mem(void)
int unhook(void)
void CRC_AddByte(unsigned short *pCrc, unsigned char data)
void CRC_AddWord(unsigned short *pCrc, unsigned short data)
void CRC_Add3Bytes(unsigned short *pCrc, unsigned long data)
void CRC_AddLong(unsigned short *pCrc, unsigned long data)
void CRC_AddByteArray(unsigned short *pCrc, unsigned char* data, int size)
void CRC_AddString(unsigned short *pCrc, unsigned char* str)
void CRC_ResetCRC(unsigned short *pCrc, unsigned short seed)
unsigned short CRC_GetCRC(unsigned short *pCrc)
```

9.6.1 初始 hook（等待 MCU 复位）

PC 的所有初始化结束后，将会进入循环以等待来自 MCU 的通信。int `hook_reset(void)` 例程实施与 MCU 建立初始通信的所有必要步骤。

9.6.2 检查 ACK

例程 int `could_be_ack(unsigned b)` 会根据通信速率不同其所能接收的全部字符集来判断接收到的字符是否符合要求。

9.6.3 速度校准

int `calibrate_speed(void)` 例程中实施的速度校准循环执行从机频率校准中所述的操作。如果未接收到来自 MCU 的 ACK，则发送另一个 break 字符。

9.6.4 读取 MCU 信息

成功完成校准后，PC 立即请求 `Ident` 命令，MCU 以它自身的信息进行响应。这通过 int `read_mcu_info(void)` 函数实现。

9.6.5 映像操作

函数 int `setup_vect_tbl(void)` 和 int `check_image()`，如 8 位和 32 位 MCU 映像操作所述。

9.6.6 块操作

执行三个主要的数据交换操作：

- 擦除块
- 读块
- 写（编程）块

这些基本操作由以下函数完成：

```
int erase_blk(unsigned a)
int read_blk(unsigned adr, int len, BYTE *dest)
int prg_blk(unsigned a, int len)
```

实际实施比较简单，遵循的是解释 MCU 命令中所述的规则。

9.6.7 主程序循环

引导加载程序编程算法是在函数 int `prg_area(unsigned start, unsigned end)` 中实施的。该函数的功能是从映像中读取数据，并将其分成适当大小的块（最小擦除 / 写块的大小）。然后调用擦除块例程和写块例程。

该例程还将过程信息输出到标准的 I/O（如块边界地址、进程指示）。

还包括另一个辅助函数 `int prg_mem(void)`。它可以获得必须被编程的最低和最高存储地址，因为调用 `int prg_area(unsigned start, unsigned end)` 函数时需要用到这些地址。

9.6.8 CRC 计算

新版引导加载程序协议添加了通过 CRC-CCITT 检查保护串行通信协议的选项。因此，PC 源代码包含了一些简单的功能，以支持这种可能性：

```
void CRC_AddByte(unsigned short *pCrc, unsigned char data)
void CRC_AddWord(unsigned short *pCrc, unsigned short data)
void CRC_Add3Bytes(unsigned short *pCrc, unsigned long data)
void CRC_AddLong(unsigned short *pCrc, unsigned long data)
void CRC_AddByteArray(unsigned short *pCrc, unsigned char* data, int size)
void CRC_AddString(unsigned short *pCrc, unsigned char* str)
void CRC_ResetCRC(unsigned short *pCrc, unsigned short seed)
unsigned short CRC_GetCRC(unsigned short *pCrc)
```

9.6.9 最终 unhook

函数 `int unhook(void)` 发出退出命令。

10 主机应用程序用户指南

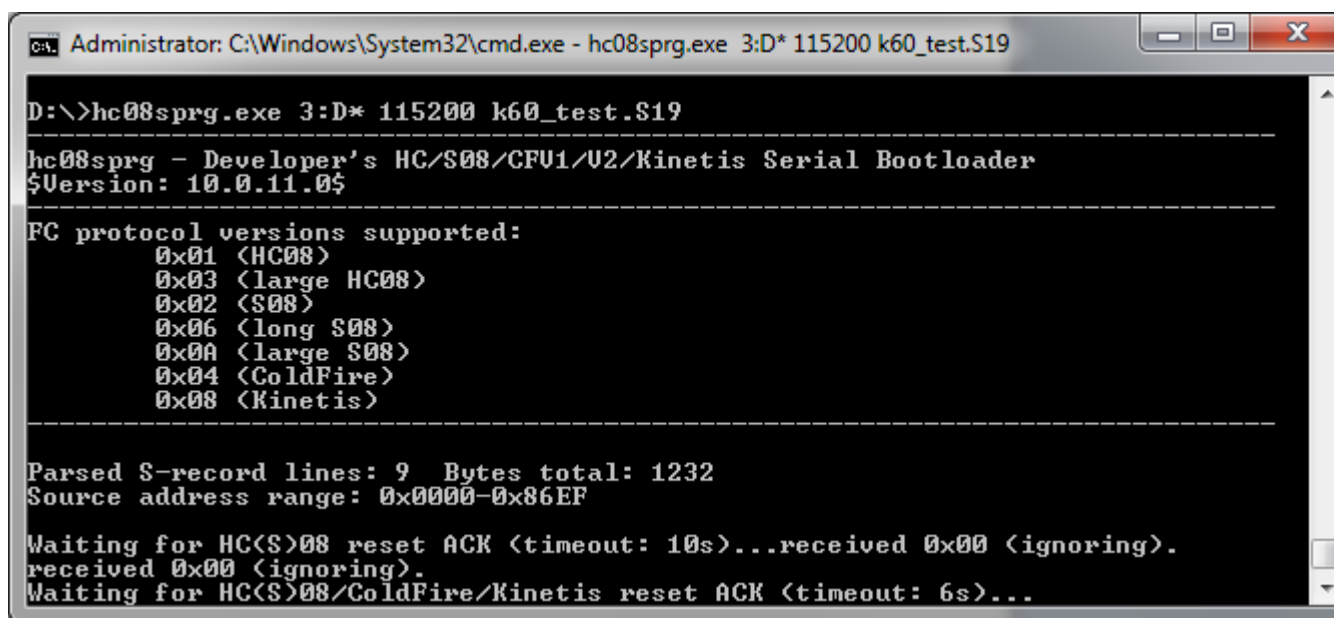
像其他任何 8 位 MCU 一样将引导加载程序二进制代码 (S19 文件) 下载到 MCU 中 (使用 MON08 串行编程器或其他，HCS08 使用 BDM 接口)。然后，在应用中插接或焊接 MCU。

借助预编程到 MCU 中的引导加载程序，用户可以使用引导加载程序实用程序通过 SCI 接口下载 8 位 MCU 用户应用代码。

10.1 引导加载操作 (命令行版本)

在 Linux 中打开一个命令提示符或 Windows 目录，复制 hc08sprg 可执行文件和 S19 文件。

例如，假设串口板连接到第二个串口 (COM3/dev/ttyS1，速度设置为 115200 bd/s 且使用 short trim) 且尚未上电，则会使用以下序列：`hc08sprg.exe 3:D* 115200 k60_test.s19` 调用引导程序。



```
Administrator: C:\Windows\System32\cmd.exe - hc08sprg.exe 3:D* 115200 k60_test.S19
D:\>hc08sprg.exe 3:D* 115200 k60_test.S19
-----
hc08sprg - Developer's HC/S08/CFU1/U2/Kinetis Serial Bootloader
$Version: 10.0.11.05
-----
PC protocol versions supported:
    0x01 (HC08)
    0x03 (large HC08)
    0x02 (S08)
    0x06 (long S08)
    0x0A (large S08)
    0x04 (ColdFire)
    0x08 (Kinetis)
-----
Parsed S-record lines: 9 Bytes total: 1232
Source address range: 0x0000-0x86EF
Waiting for HC(S)08 reset ACK (timeout: 10s)...received 0x00 (ignoring).
received 0x00 (ignoring).
Waiting for HC(S)08/ColdFire/Kinetis reset ACK (timeout: 6s)...
```

图 38. 引导加载程序调用

现在引导加载程序期待接收到来自 MCU 引导加载程序使能应用的 ACK 命令。然后开启串口板电源，如果所有的连接都正常，则 MCU 将开始与 PC 通信。如果 IDENT 命令后使用的是已知通信速度的引导加载程序版本，则其不会执行校准程序。从 MCU 获取的信息随后将显示在屏幕（图 39）上。

```

Administrator: C:\Windows\System32\cmd.exe - hc08sprg.exe 1:D* 115200 k60_test.S19

D:\>hc08sprg.exe 1:D* 115200 k60_test.S19
-----
hc08sprg - Developer's HC/S08/CFU1/U2/Kinetis Serial Bootloader
$Version: 10.0.11.05
-----
FC protocol versions supported:
    0x01 (HC08)
    0x03 (large HC08)
    0x02 (S08)
    0x06 (long S08)
    0x0A (large S08)
    0x04 (ColdFire)
    0x08 (Kinetis)
-----

Parsed S-record lines: 9 Bytes total: 1232
Source address range: 0x0000-0x86EF

Waiting for HC(S)08/ColdFire/Kinetis reset ACK (timeout: 8s)...received 0x00 (ignoring).
Waiting for HC(S)08/ColdFire/Kinetis reset ACK (timeout: 7s)...received 0xfc (good).
Calibration break pulse sent. Count: 3

Bootloader protocol version: 0x08 (Kinetis, read command supported, Protocol secured: CRC-CCITT)
Bootloader version string: K60
System device ID: 0x14A [Kinetis K60] rev. 0
Kinetis Package: 144-pin
Number of memory blocks: 1
Memory block #1: 0x00004000-0x0007FFFF
Erase block size: 2048 bytes
Write block size: 128 bytes
Original vector table: 0x00000000-0x000003FF
New vector table: 0x00004000-0x000043FF

Are you sure to program part? [y/N]:

```

图 39. 第一阶段引导加载

输入 ‘y’ 确定，引导加载（FLASH 重编程）将继续。用户应用程序随后会启动。


```

Administrator: C:\Windows\System32\cmd.exe
D:\>hc08sprg.exe 3:D* 115200 k60_test.S19
-----
hc08sprg - Developer's HC/S08/CFU1/U2/Kinetis Serial Bootloader
$Version: 10.0.11.0$
-----
FC protocol versions supported:
  0x01 (HC08)
  0x03 (large HC08)
  0x02 (S08)
  0x06 (long S08)
  0x0A (large S08)
  0x04 (ColdFire)
  0x08 (Kinetis)
-----
Parsed S-record lines: 9 Bytes total: 1232
Source address range: 0x0000-0x86EF
Waiting for HC(S)08/ColdFire/Kinetis reset ACK (timeout: 7s)...received 0x00 (ignoring).
received 0xfc (good).
Send Standard 0.
Calibration break pulse sent. Count: 1
Bootloader protocol version: 0x08 (Kinetis, read command supported, Protocol secured: CRC-CCITT)
Bootloader version string: K60
System device ID: 0x14A [Kinetis K60] rev. 0
Kinetis Package: 144-pin
Number of memory blocks: 1
Memory block #1: 0x00004000-0x0007FFFF
Erase block size: 2048 bytes
Write block size: 128 bytes
Original vector table: 0x00000000-0x000003FF
New vector table: 0x00004000-0x000043FF
Are you sure to program part? [y/N]: y
Memory is erased.
Memory programmed:          100%
Memory verified:            OK
D:\>

```

图 40. 引导加载已完成

10.1.1 内存边界重叠实例

如果用户尝试引导加载不适合实际 MCU 内存的应用程序，则会显示警告。用户可以决定继续，但某些存储器位置很可能会被错误编程（用户代码要么不在可用的 FLASH 存储器范围内，要么与引导代码重叠）。

```

Administrator: C:\Windows\System32\cmd.exe - hc08sprg.exe 3:D* 115200 wontFit.S19
D:\>hc08sprg.exe 3:D* 115200 wontFit.S19
-----
hc08sprg - Developer's HC/S08/CFU1/U2/Kinetis Serial Bootloader
$Version: 10.0.11.0$
-----
FC protocol versions supported:
    0x01 (HC08)
    0x03 (large HC08)
    0x02 (S08)
    0x06 (long S08)
    0x0A (large S08)
    0x04 (ColdFire)
    0x08 (Kinetis)
-----
Parsed S-record lines: 113 Bytes total: 3054
Source address range: 0x0000-0x0FE5

Waiting for HC(S)08 reset ACK (timeout: 10s)...received 0x00 (ignoring).
received 0xfc (good).
Send Standard 0.
Calibration break pulse sent. Count: 1

Bootloader protocol version: 0x08 (Kinetis, read command supported,
Protocol secured: CRC-CCITT)
Bootloader version string: K60
System device ID: 0x14A [Kinetis K60] rev. 0
Kinetis Package: 144-pin
Number of memory blocks: 1
Memory block #1: 0x00004000-0x0007FFFF
Erase block size: 2048 bytes
Write block size: 128 bytes
Original vector table: 0x00000000-0x000003FF
New vector table: 0x00004000-0x000043FF

WARNING! S19 image will not fit into available memory (at address 0x0000400)!
Are you sure to program part? [y/N]:

```

图 41. 内存边界重叠实例

10.2 引导加载操作（windows 版）

此外，还存在（基于相同源代码库）Windows 用户友好型 PC 主机应用程序版本。该应用程序允许通过引导加载程序执行单独步骤，还允许通过命令行版本执行自动步骤。

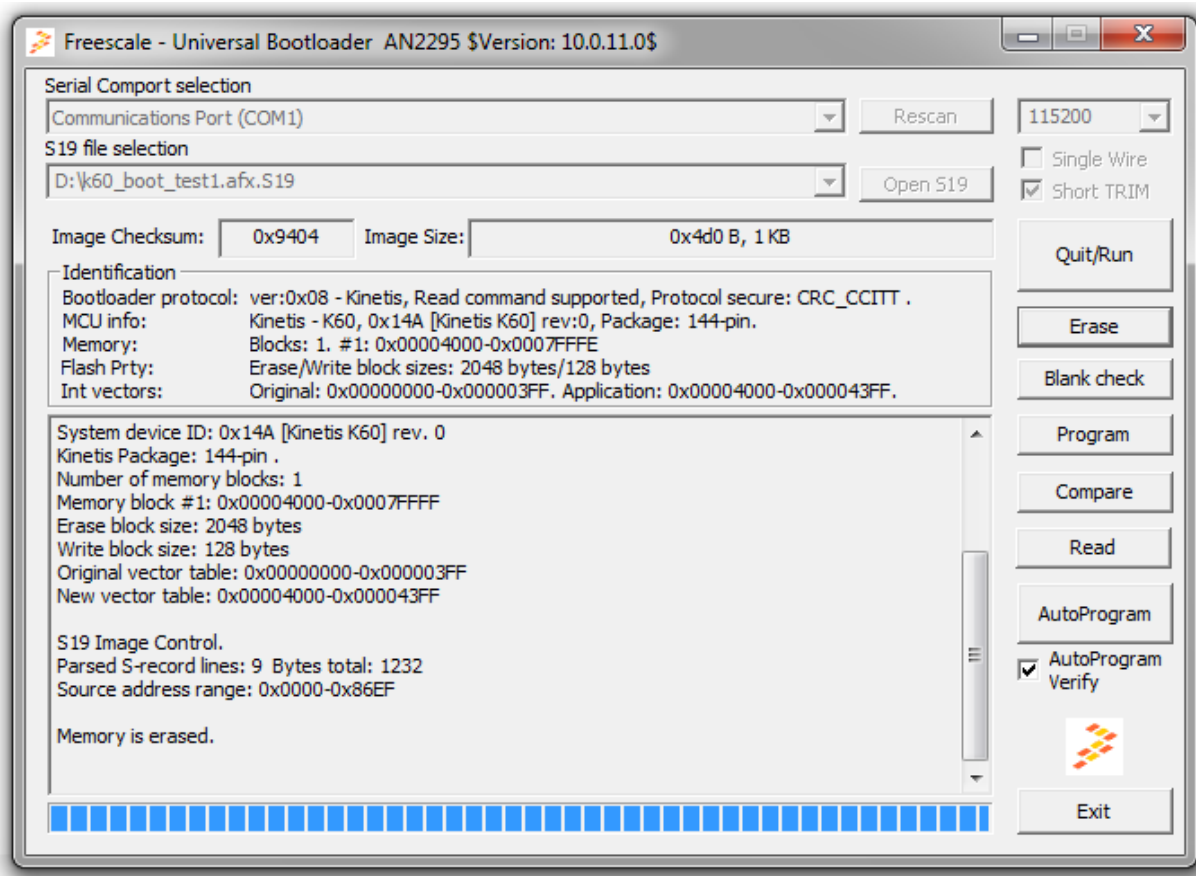


图 42. 基于 Windows 的 PC 主机应用程序

10.2.1 如何使用 Windows 版的主机应用程序

10.2.1.1 打开 “win_hc08sprg.exe”

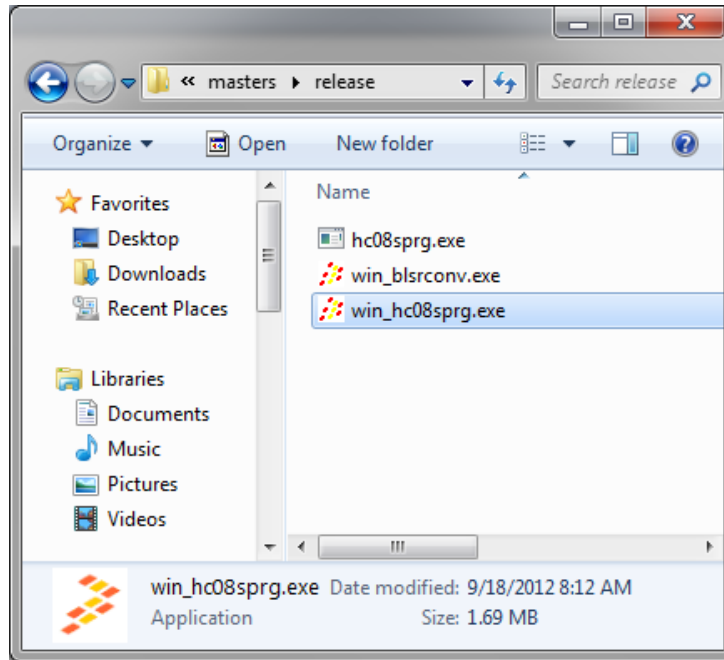


图 43. PC 主机释放文件夹

这存储在 PC 主机应用程序的 release 文件夹中。

10.2.1.2 设置应用程序以与目标连接

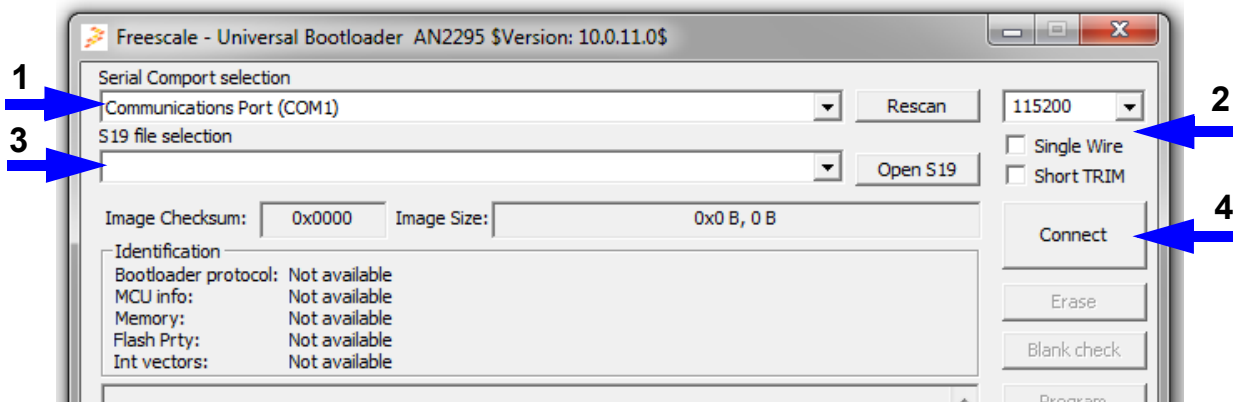


图 44. 设置应用程序以与目标连接

1. 选择正确的通信端口。如果端口不在列表中，则尝试通过点击按钮 “Rescan” 重新扫描 PC 中的通信端口。
2. 设置通信选项以满足目标设置。
 - 通信速度 - 选择通信波特率
 - Single Wire - 检查目标是否通过单线连接连接

- Short TRIM - 检查是否已将目标配置为已使用的短时钟校准（调整）脉冲。
- 3. 选择 S19 文件 - 要将新 S19 文件添加到列表中，请使用“Open S19”按钮，要再次使用已打开的任何文件，只需从组合框中选择它。
- 4. 连接目标 - 点击“Connect”并通过已启用的引导加载程序启动选项运行目标。

10.2.1.3 使用 Windows 版的主机应用程序

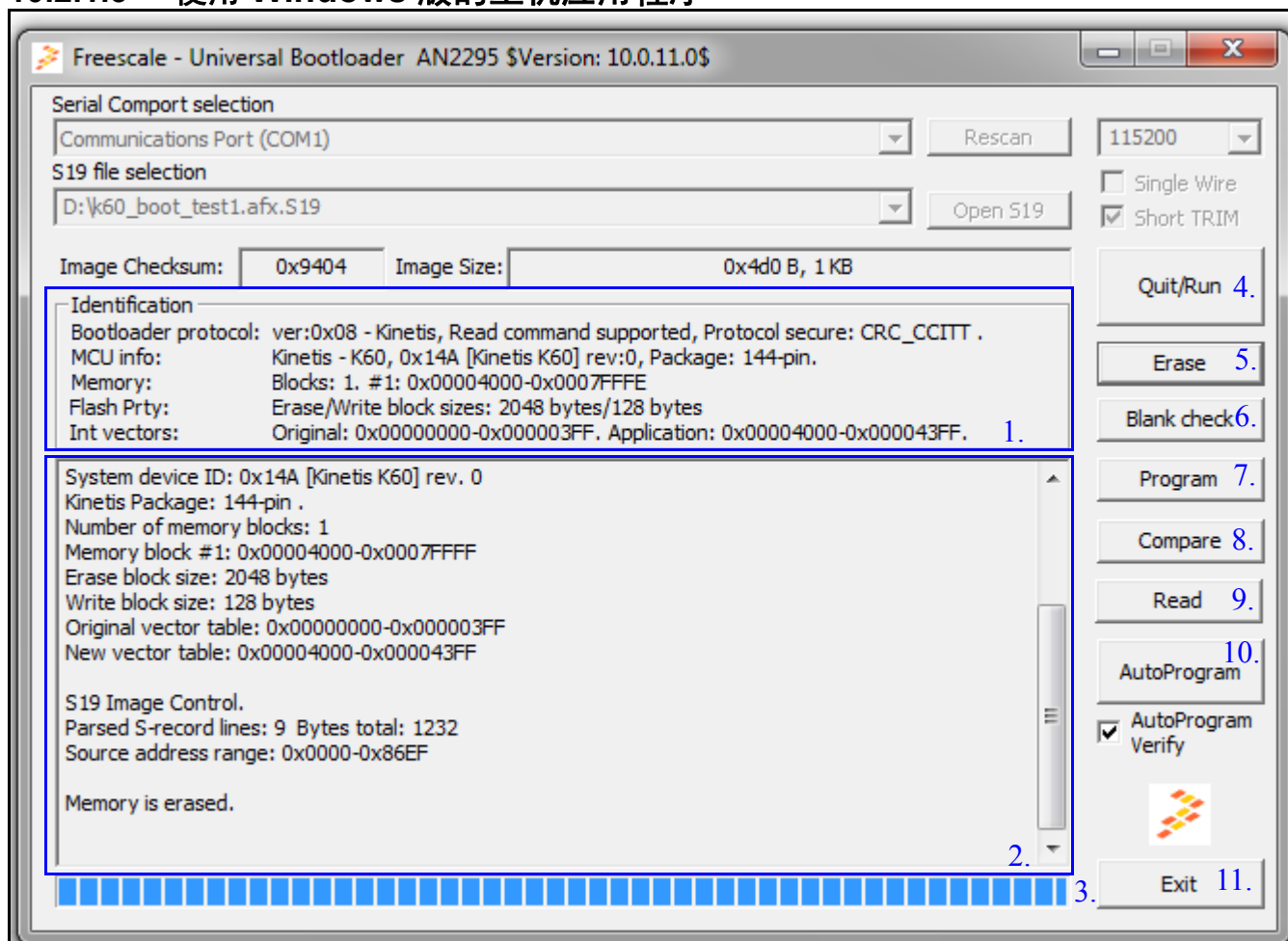


图 45. 已连接至目标的 Windows 版引导加载程序窗口

1. 识别信息 - 在此框架中，应用程序将显示与目标和已使用 AN2295 串行协议相关的所有已知信息。
2. 控制台窗口 - 此控制台窗口显示的信息与应用程序的命令行版本完全相同。
3. 进度条 - 进度条显示当前活动操作的状态（擦除、编程、比较等）。
4. 退出 / 运行 - 点击此按钮可结束当前的引导加载程序会话和调用并在可用时运行用户应用程序。
5. 擦除 - 点击此按钮可调用擦除整个用户 Flash 内存区。
6. 空白检查 - 此命令可以检查是否已擦除内存。此命令由读命令实施。

7. 编程 - 点击此按钮可尝试将准备的映像下载至目标。执行此操作之前，请确保已擦除目标内存。
8. 比较 - 点击此按钮可将目标内存的内容与准备的映像作比较。
9. 读取 - 此命令可读取整个目标用户区域并将其存储至新的 S19 文件。
10. AutoProgram - 此命令可调用与使用 AN2295 主机应用程序的命令行版本完全相同的程序。要在写入映像后添加额外的验证，则必须选中 AutoProgram Verify。
11. 退出 - 点击此按钮可退出引导加载程序应用程序

11 合并引导加载程序 and 应用程序映像

本节详细介绍将引导加载程序 and 用户应用程序 S19 文件合并到一个输出 S19 文件的计算机软件，采用 zip 格式的该软件可从飞思卡尔半导体公司主页 freescale.com 下载。所有代码均用 C 语言编写，兼容 Win32[®] 平台。

合并引导加载程序 and 用户应用程序 S19 文件的典型事例为通过引导加载程序的功能最终量产 MCU Flash 映像。通过调试闪存引导加载程序的引导加载应用程序的接口，此工具可以简化闪存引导加载程序本身的操作。

合并工具应用使用与所有其他 AN2295 PC 软件相同的源代码库（主机引导加载程序应用程序：命令行和标准 Windows Form 版本）。

合并工具 Windows Form 应用程序用于加载两个 S19 文件（应用程序 and 引导加载程序），并且可以选择 MCU 类型 and 中断向量表（原始 and 重定位）。该应用程序还包含日志窗口，用于提供与合并流程相关的详细信息 and 所有可能的警告。

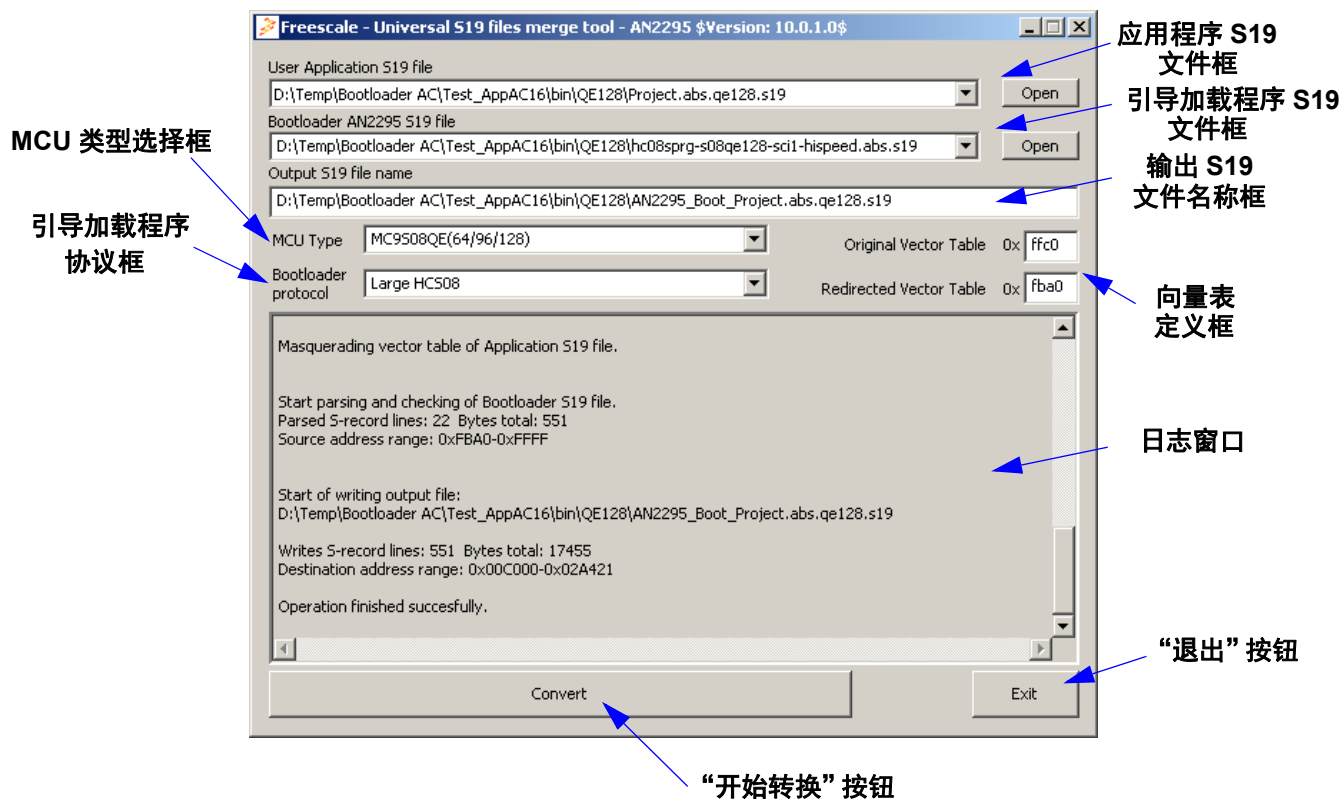


图 46. AN2295 S19 合并工具

12 参考文献

如需了解更多信息，请参考飞思卡尔半导体主页 freescale.com

- AN2295SW: 包含采用 zip 格式、适合于该应用笔记的所有软件文件。
- HCS08RMv1: *HCS08 Family Reference Manual Volume 1*
- AN1831: *Using MC68HC908 On-Chip FLASH Programming Routines*
- AN2140: *Serial Monitor for MC9S08GB/GT*
- AN2498: *Initial trimming of the MC68HC908 ICG*
- AN2504: *On-Chip FLASH Programming API for CodeWarrior Software*
- AN2508: *Generating Clocks for HC908 MCU Families*
- AN2545: *On-Chip FLASH Programming Routines for MC68HC908GR/GZ*
- AN2637: *Software SCI MC68HC908QT/QY MCU*
- AN2635: *On-Chip FLASH Programming Routines for LB8 and other FLASH-based MCUs*
- AN2874: *Using M68HC908 ROM-Resident Routines*
- AN3153: *Using the Full-Speed USB Module on the MCHC908JW32*

参考文献

- ZSTARRM: *Wireless Sensing Triple Axis Reference design*
- CFPRM: *ColdFire® Family Programmer's Reference Manual*
- K60P100M100SF2RM: K60 子系列参考手册
- AN3942: *Flash Programming Routines for the HCS08 and the ColdFire (V1) devices*
- 主机应用程序用户指南 节 10, 主机应用程序用户指南
- Kinetis 协议版本描述中包括以下用户应用程序更改: 节 7, FC 协议, 版本 5, Kinetis
- 修改 Kinetis 用户应用程序以准备执行 AN2295 引导加载程序入门指南: 节 7.8, 快速指南: 如何为 AN2295 引导加载程序准备 Kinetis 用户应用

How to Reach Us:

Home Page:

freescale.com

Web Support:

freescale.com/support

本文档中的信息仅供系统和软件实施方使用 Freescale 产品。本文并未明示或者暗示授予利用本文档信息进行设计或者加工集成电路的版权许可。Freescale 保留对此处任何产品进行更改的权利，恕不另行通知。

Freescale 对其产品在任何特定用途方面的适用性不做任何担保、表示或保证，也不承担因为应用程序或者使用产品或电路所产生的任何责任，明确拒绝承担包括但不限于后果性的或附带性的损害在内的所有责任。Freescale 的数据表和 / 或规格中所提供的“典型”参数在不同应用中可能并且确实不同，实际性能会随时间而有所变化。所有运行参数，包括“经典值”在内，必须经由客户的技术专家对每个客户的应用程序进行验证。Freescale 未转让与其专利权及其他权利相关的许可。Freescale 销售产品时遵循以下网址中包含的标准销售条款和条件：

freescale.com/SalesTermsandConditions.

Freescale, the Freescale logo, and Kinetis, are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners.

© 2011 Freescale Semiconductor, Inc.

© 2011 飞思卡尔半导体有限公司

Document Number: AN2295
Rev. 13, 10/2013



