Freescale Semiconductor
应用笔记

Document Number: AN4759
Rev. 0, 06/2013

在MC56F84xxx DSC系列器件上实现引导加载程序

作者： Xuwei Zhou

1 简介

除了应用代码之外，许多应用程序还需要一段称之为引导加载程序的代码驻留在非易失性存储器中。引导加载程序与应用代码完全无关，其主要功能就是与主机通信，以获取更新的应用代码，并在芯片上的非易失性存储器的应用代码区域中进行代码编程。在MC56F84xxx数字信号控制器（DSC）系列中，Flash为非易失性存储器，它不同于早期DSC系列中的Flash存储器。

本应用笔记旨在以MC56F84789为例，详细解释了如何在MC56F84xxx系列DSC上实现引导加载程序。本文档将装有Processer Expert（PE）的CodeWarrior10.3用作开发环境。

目录

© 2013 Freescale Semiconductor, Inc.

2 引导加载程序机制

复位系统后，引导加载程序代码将首先执行，并决定是否要为Flash存储器重新编程。如果要为Flash重新编程，系统将与某台主机通信，以接收并更新应用代码。完成更新后，将启动更新的应用代码。如果不需要更新，从引导加载程序退出后将执行当前的应用代码。

2.1 MC56F84789 DSC中的存储器映射

详细了解MC56F84789中的存储器映射是非常有必要的。在**freescale.com**上提供的*MC56F847XXRM：MC56F847xx参考手册*第4章：“存储器映射”中可以找到这些信息。对于MC56F84789，芯片中有两块Flash存储器和两块RAM。可以通过程序存储器（program memory bus）总线和数据存储器总线（data memory bus）存取这两块Flash存储器和其中的一块RAM。也就是说，在这3个存储器中，每个存储器都有两组地址分别映射到程序存储器空间和数据存储器空间中。图1显示了MC56F84789的存储器映射。

注意：将这两块Flash存储器称为“程序（program）”和“数据（data）”Flash只是一种惯例。

程序Flash （Program flash）：

- 其容量为128 K字，相对较大。
- 该Flash从程序存储器映射中的地址0x0000开始，因此更适合用于存储代码。
- 由于它还会映射到图1中所示的数据存储器空间，因此也可用于存储包含常量值的变量。

数据Flash （Data flash）：

- 数据Flash采用了相同的命名规则；之所以称为“数据”Flash，是因为其容量为16 K字，相对较小。
- 该Flash从数据存储器映射中的0x8000开始，因此更适合用于存储包含常量值的变量。
- 由于它还会映射到程序存储器空间，因此也可用于存储代码。

有16 K字的RAM同时映射到了程序存储器空间和数据存储器空间，因此，代码也可以在其中运行。FlexRAM只会映射到数据存储器空间。可以将它当作传统的RAM进行存取，或者将它连同一部分数据Flash配置为增强型EEPROM。有关EEPROM在MC56F84xxx上的使用情况，请参见**freescale.com**中的*AN4689：MC56F84xxx DSC上的EEPROM*。

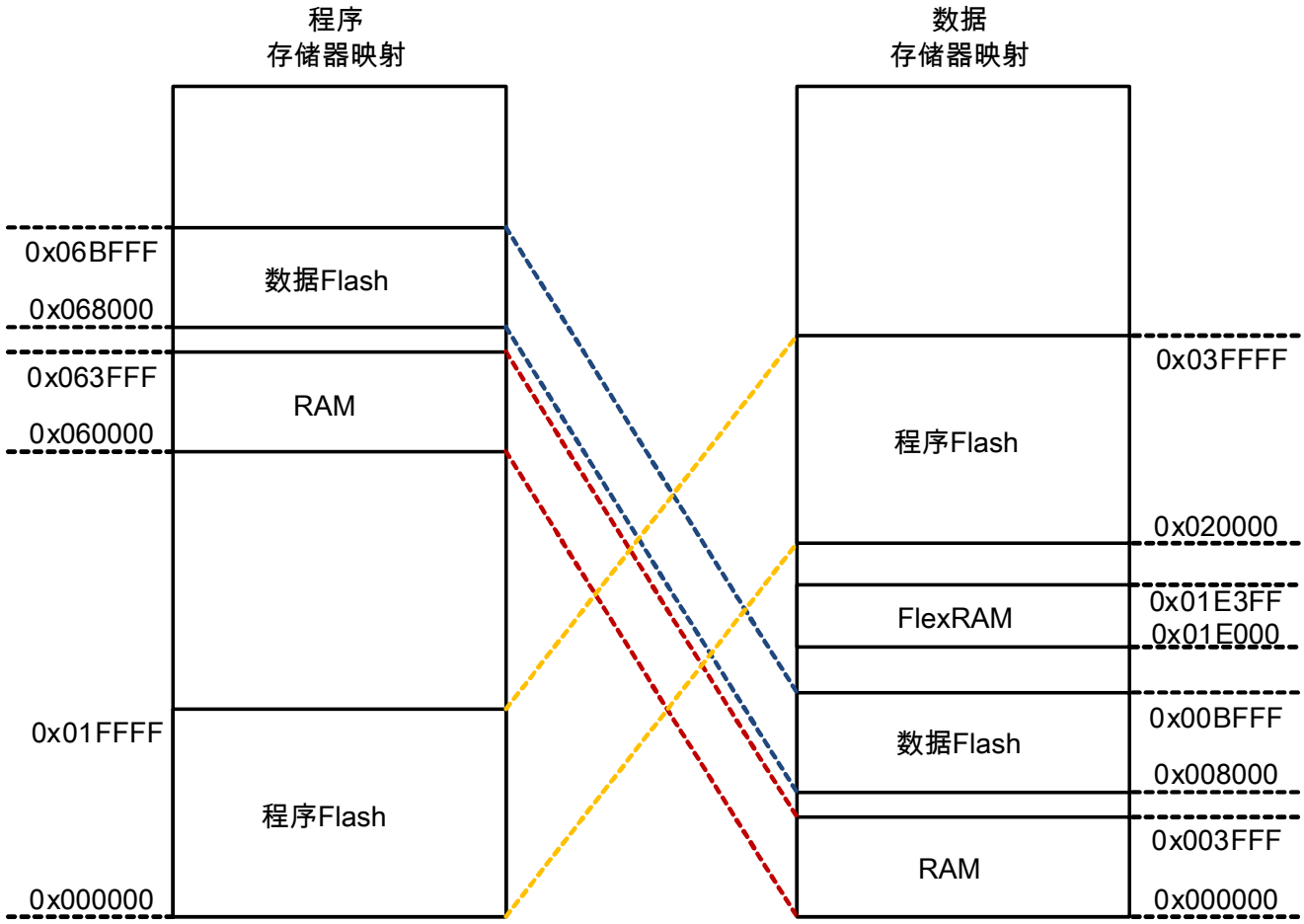

图 1. MC56F84789 DSC 中的存储器映射

2.2 引导加载程序的存储器配置

在本应用中，引导加载程序位于程序Flash的顶部并占用3 K字，而用户应用代码位于程序Flash的其余部分中。至于是要使用数据Flash来运行代码还是存储变量，完全由用户应用决定。图2显示了引导加载程序的存储器配置方式。

- 程序Flash中的0x1F400–0x1FFFF用于存储引导加载程序代码，以及全局变量的初始值（即.data段）。引导加载程序从地址0x1F400开始。
- 程序Flash中的0x1F3FD–0x1F3FF用于存储用户应用的起始地址以及延迟时间值。
- 程序Flash中的0x00000–0x1F3FC保留给用户应用代码使用。

RAM已通过链接器文件分割成两部分：

- 包含低地址的部分将映射到数据存储器空间0x0000–0x1FFF，因此用于存储变量和堆栈
- 包含高地址的部分将映射到程序存储器空间0x62000–0x63FFF；用于运行擦除/程序Flash期间可能会执行的某些函数

以下是与存储器配置有关的引导加载程序运行步骤。

1. 复位MCU后，PC寄存器会指向包含一个跳转指令的复位向量地址。
2. MCU将跳转到引导加载程序的起始地址（即0x1F400），然后，引导加载程序将开始运行。
3. SCI用来与主机通信，因此，启动引导加载程序后，将监视SCI端口。
4. 监视将持续几秒钟，如果在监视期间未收到有效数据，则会跳转到0x1F3FD–0x1F3FE中存储的地址。0x1F3FF中存储的数据确定了这种监视将要持续的秒数。
    - 对于已完成更新的MCU，0x1F3FD–0x1F3FE中存储的32位地址值必须是用户应用代码的起始地址。
    - 对于不包含应用代码，而只包含引导加载程序代码的MCU，该存储地址应该是引导加载程序的起始地址；在本例中，0x1F400为引导加载程序的起始地址。

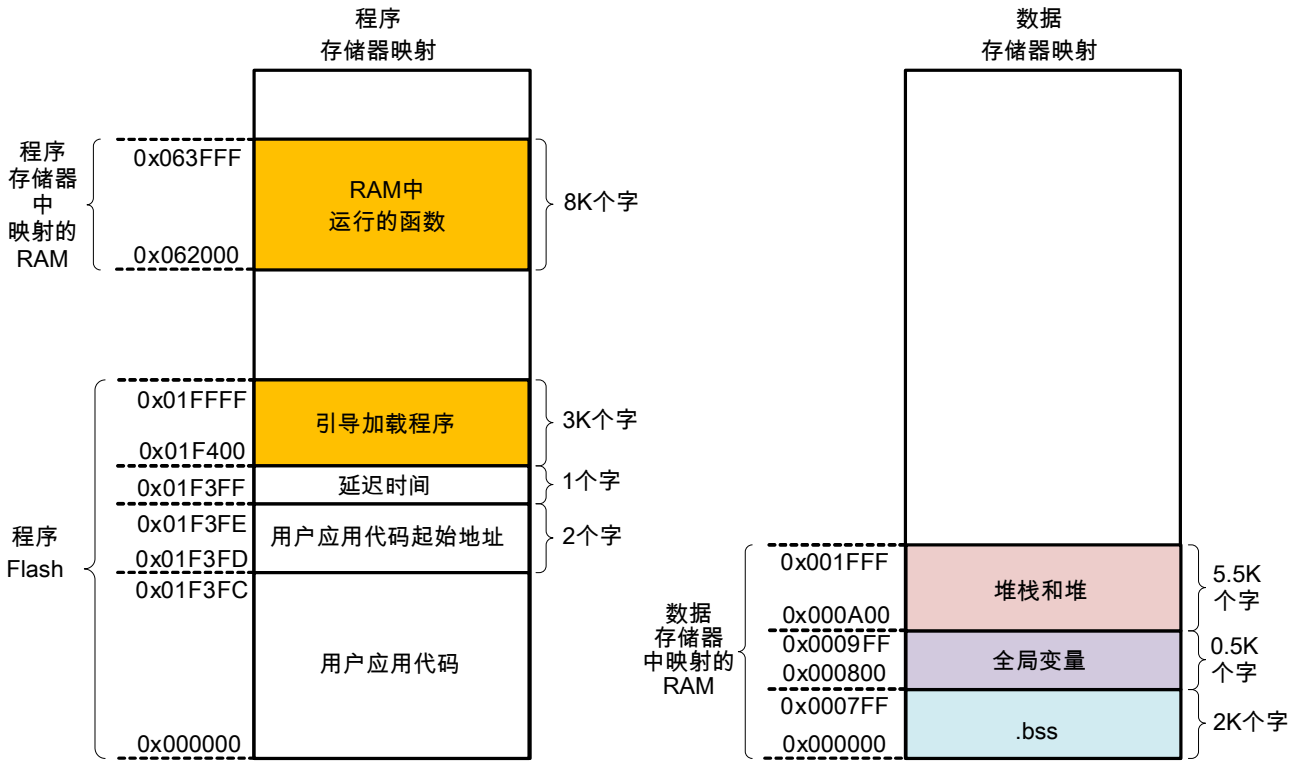

图 2. MC56F84789 中引导加载程序的存储器配置

图3显示了引导加载程序的执行流程。如果在监视期间收到有效数据，则会继续运行引导加载程序，同时擦除和编程不用来存储引导加载程序的程序Flash和数据Flash （不包括程序Flash中的0x1F400–0x1FFFF）。

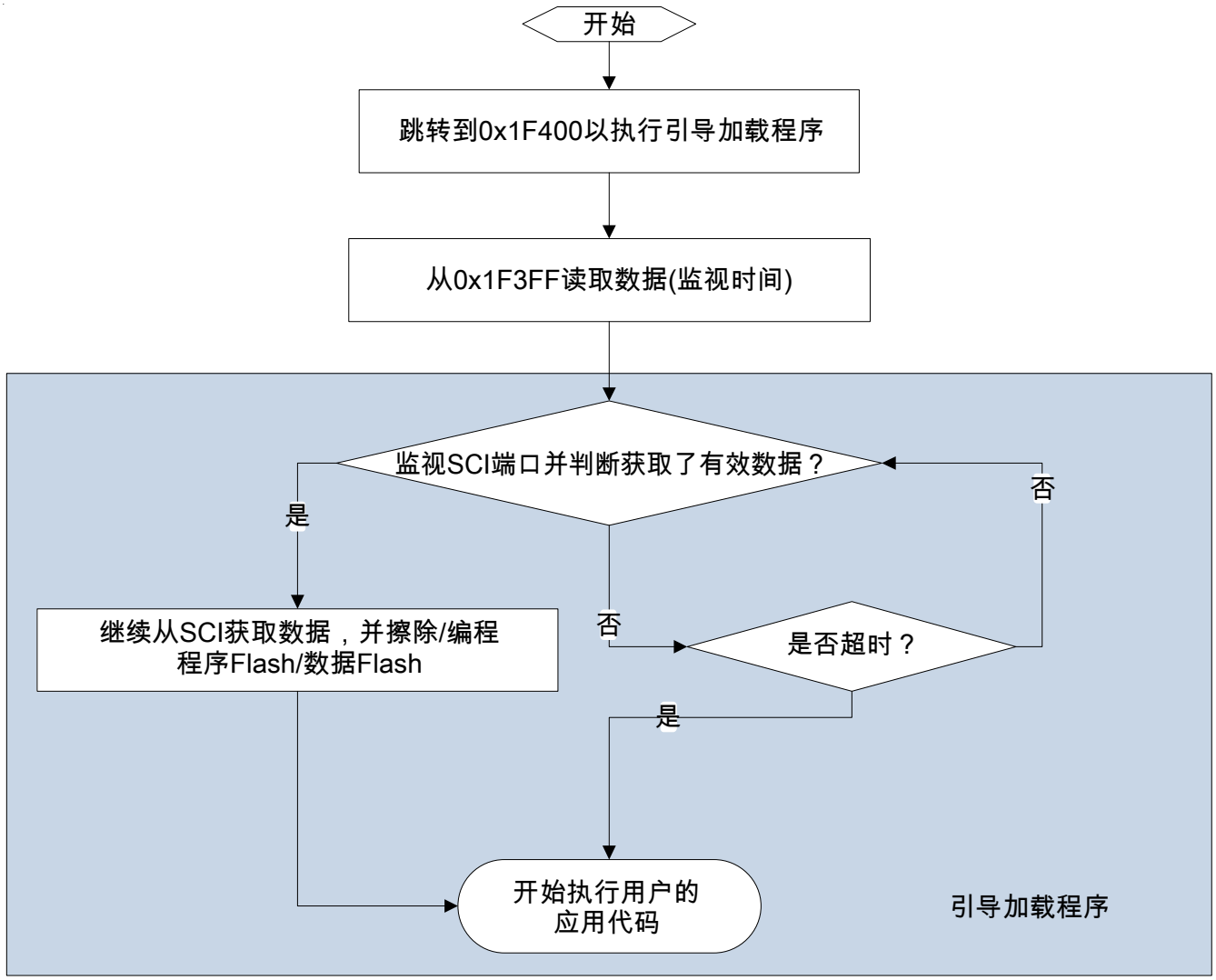

图 3. 引导加载程序流程图

3 引导加载程序的实现

如前所述，实现引导加载程序的基本思路是从主机（通常是PC）中获取应用代码，然后将其写入Flash存储器。因此，必须知道应用代码在PC中的存储方式，以及Flash擦除/编程的方法。

3.1 DSC系列的S-record格式

如果一个工程在CodeWarrior中编译链接没有错误，则可以生成s-record文件。s-record文件以扩展名“.s”结尾。此文件中的内容遵循s-record格式，这是一种可打印的格式，用于对代码中的程序和数据进行编码，使其能够在计算机系统之间传输。此文件包含一些记录（records或s-record），而这些记录基本上是由标识记录类型、记录长度、存储地址、代码/数据以及校验和的多个域构成的字符串。

二进制数据的每个字节编码为双字符十六进制数：第一个字符代表字节的高阶4位，第二个字符代表字节的低阶4位。针对MC56F84xxx DSC系列会生成3种类型的s-record：S0（类型 0）、S3（类型 3）和S7（类型 7）。以下是s-record文件的大致内容。必须注意，所有记录以ASCII代码格式存储在该文件中。

```
S0110000000050524F4752414D264441544196
S3150000000054E100F454E100F454E22A0254E22A02D4
S3150000000854E22A0254E22A0254E22A0254E22A025A
S3150000001054E22A0254E22A0254E22A0254E22A0252
S3150000001854E22A0254E22A0254E22A0254E22A024A
...
S30B0000F3FD310200000C00C5
S70500000231C7
```

下图以其中一条s-record为例，以标签的形式解释了记录的组成。

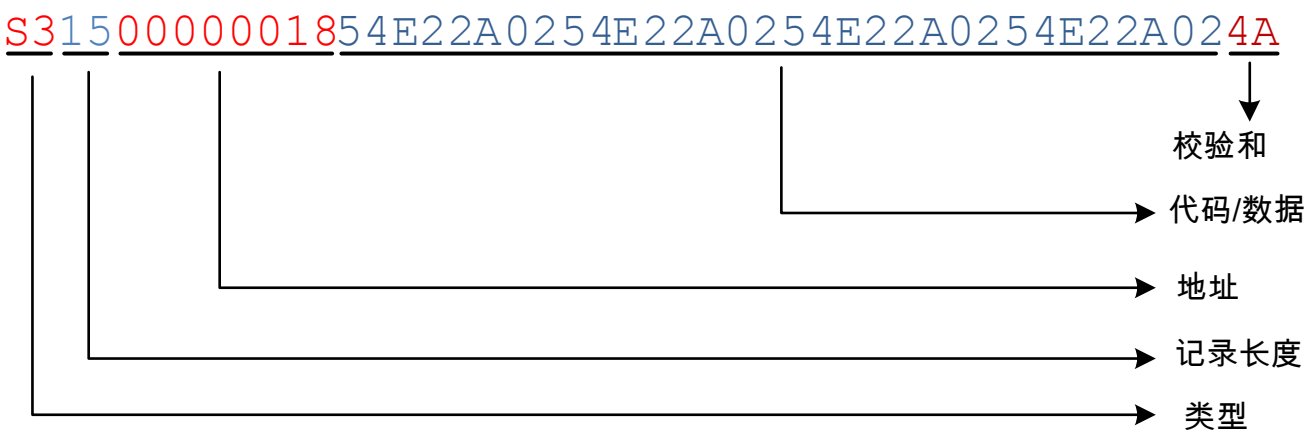

图 4. 解析 s-record

- **类型**：有8种类型s-record，用于配合编码、传输和解码功能。但此处只会生成下列3种类型的s-record。
  - **S0**：这是每个 s-record 文件中的第一条记录。代码/数据域可以包含任何描述性信息用于标识该文件中后续的 s-record。地址域通常为 0。DSC 系列器件中生成的 S0 记

录为：
S011000000050524F4752414D264441544196
代码/数据域“50524F4752414D2644415441”实际上是“PROGRAM&DATA”的 ASCII 代码。

- **S3**：此记录包含代码/数据，以及代码/数据要存放的 32 位起始字地址。
- **S7**：S3 记录块的结尾记录。地址域可以有选择性地包含整个程序流程中第一条指令所在的 32 位字地址。不存在代码/数据域。
  例如，在 s-record S70500000231C7 中，地址“00000231”实际上是函数“F_EntryPoint”的起始地址。

- **记录长度**：记录中字符对（不包括类型和记录长度）的数目。
  在图4所示的s-record中，总共有0x15个字节，包括地址、代码以及校验和（将一对字符视为一个字节）。

- **地址**： 存储器中要将代码/数据域加载到的32位字起始地址。
  在图4所示的s-record中，将从程序Flash中的地址0x00000018开始放置有效的代码/数据。由于DSC系列器件中的地址只有21个有效位，因此，使用了bit 25来指示程序存储器空间或数据存储器空间。
  - 如果bit 25为0，则表示此地址为程序存储器空间地址。
  - 如果bit 25为1，则表示此地址为数据存储器空间地址。

  以 s-record S3110200001200000000000000000000000000DA 为例。
  此记录指示应将代码/数据段“000000000000000000000000”放入数据存储器空间中从地址 0x00012 开始的位置。由于 RAM 已映射到此区域并且与链接器文件的描述相连，因此，此代码/数据段包含数据而不是代码。

- **代码/数据**：由0到n个字节组成，包括可执行代码、数据或描述性信息。
  在S0记录中，它是描述性信息；在S3记录中，它是代码/数据。S7记录中没有代码/数据。
  在Flash和RAM的每个存储单元中应该有一个16位字，但在s-record的代码/数据段中，字节顺序模式为小端模式。在图4所示的s-record中，“54E22A0254E22A0254E22A0254E22A02”表示将0xE254放入地址0x000018，将0x022A放入地址0x000019，依此类推。

- **校验和**：由构成记录长度、地址以及代码/数据的所有字符对（代表一个16进制字节）相累加，对结果求反码，取最后一个字节。
  在图4所示的s-record中，校验和0x4A其实是 “0x15+0x00+0x00+0x00+0x18+0x54+ 0xE2+ ..+0x2A+0x02”的反码中最后一个字节。可以在引导加载程序中使用此校验和来检查收到的记录是否正确。

其中每条记录的末尾都有一个 EOL（End Of Line，行尾）标记。

3.2 循环缓冲区 （Circular buffer）

以下步骤描述了循环缓冲区机制。

1. S-record通过SCI从主PC传输到DSC，并存储在循环缓冲区中。
2. 引导加载程序将不断地解析该缓冲区中的内容，以识别是否收到了新的完整记录。
3. 收到新记录后，传输将暂时停止，并将记录中的代码/数据段编程到所需的区域。下图显示了该缓冲区的工作原理。

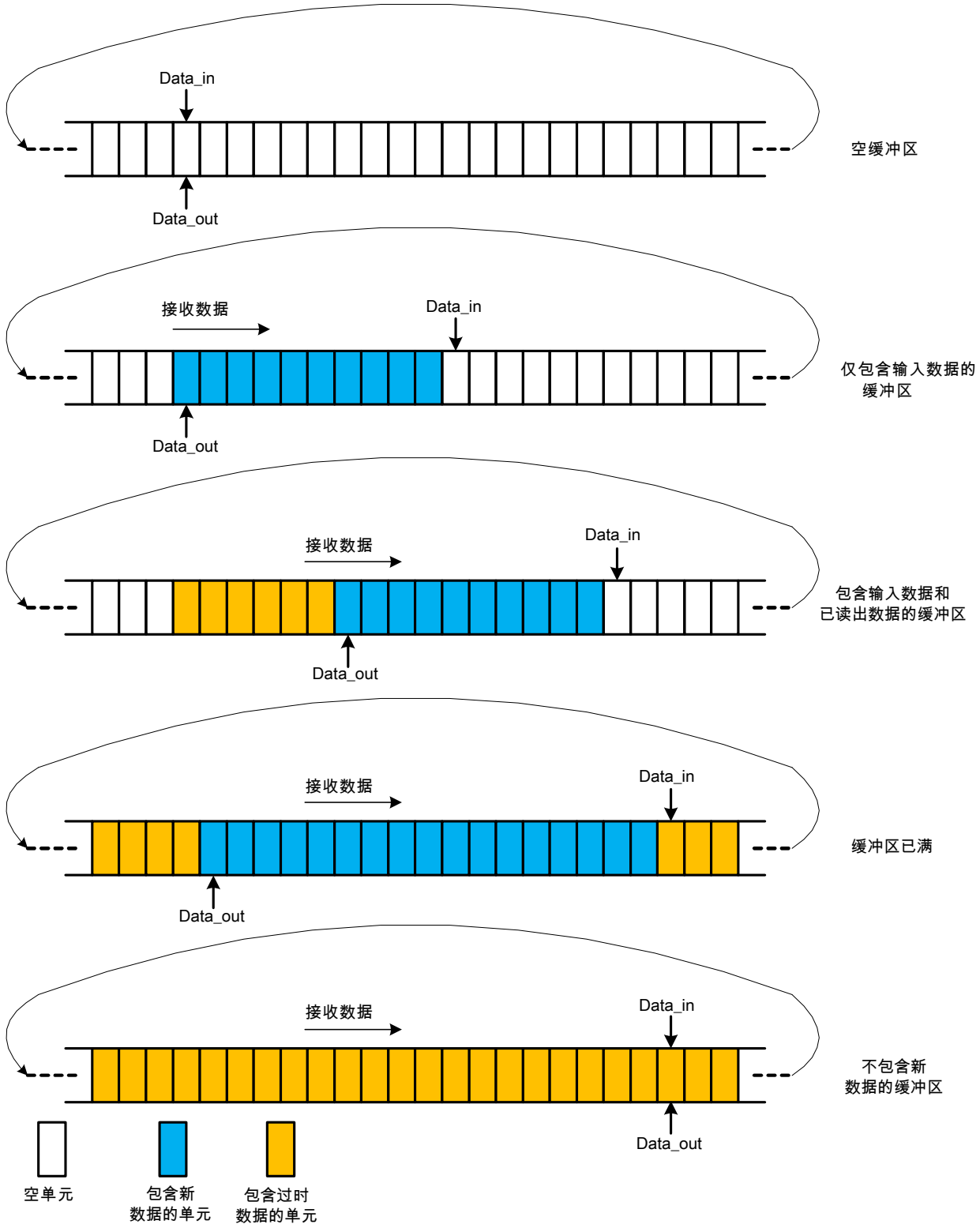
Data_in
Data_out
空缓冲区
接收数据
Data_in
Data_out
仅包含输入数据的
缓冲区
接收数据
Data_in
Data_out
包含输入数据和
已读出数据的缓冲区
接收数据
Data_in
Data_out
缓冲区已满
接收数据
Data_in
Data_out
不包含新
数据的缓冲区
空单元
包含新
数据的单元
包含过时
数据的单元

图 5. 循环缓冲区机制

使用两个指针来操作此缓冲区。

- 指针Data_in始终指向最后收到的数据即将存放的单元。
- 指针Data_out始终指向收到的第一个新数据所存放的单元。

如果这两个指针相同，则表示缓冲区中没有新数据。指针Data_in在SCI接收中断服务例程中运行，因此，SCI接收中断函数中收到的数据会自动填入至该缓冲区。必须在主循环中不断解析缓冲区中的内容，以确保接收到正确的s-record并且缓冲区不会溢出。此缓冲区是使用内核的模寻址*(*modulo addressing*)*功能实现的。有关模寻址操作的详细信息，请参见**freescale.com**上的*DSP56800E和DSP56800EX Reference Manual*。

Buffer.c和Buffer.h包含缓冲区相关的所有函数和变量。宏RX_DATA_SIZE以字节为单位定义缓冲区的大小。使用以下两个函数从缓冲区中读出数据：

- *char get_char(char **ptr)*

  此函数将读出指针 ptr 指向的数据，并将 ptr 加 1。由于 s-record 以 ASCII 格式存储，因此，*get_char(ptr)*读出的数据也是 ASCII 码。下图显示了相关工作原理。

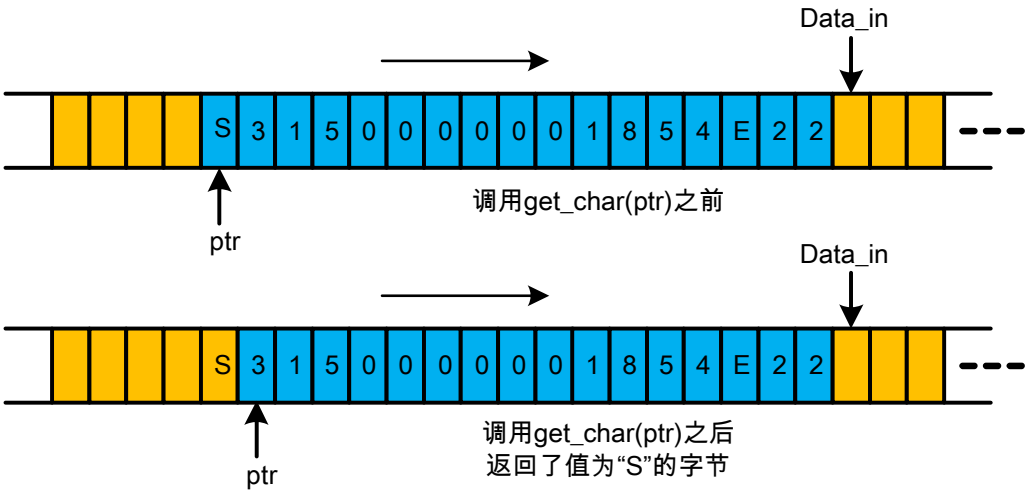

图 6. *get_char()*函数

- *char get_byte(char **ptr)*

  此函数将读出指针 ptr 指向的当前以及下一个数据，并将 ptr 加 2。首先会将两个 ASCII 数据变换为相应的整数，然后将它们串连在一起作为返回值。下图显示了相关工作原理。

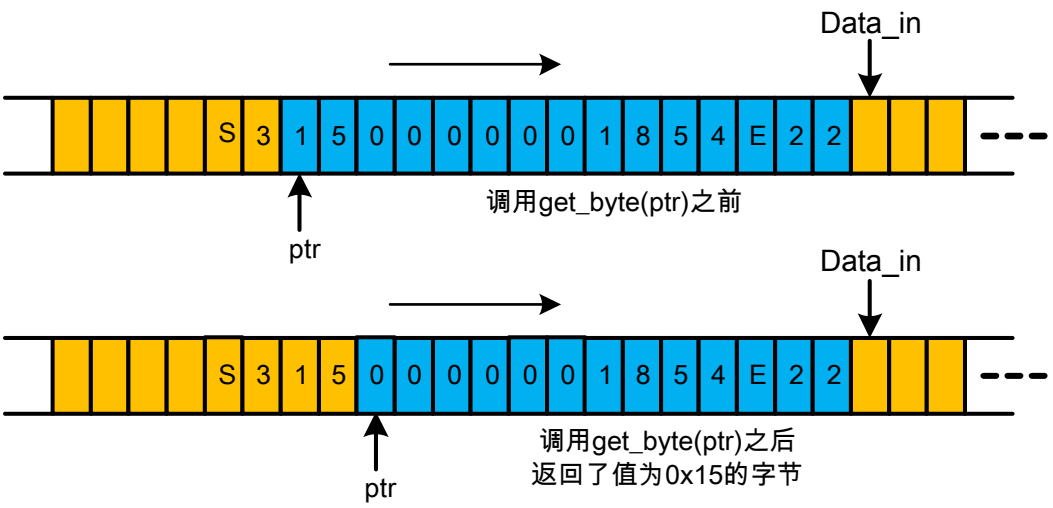

图 7. *get_byte()*函数

3.3 引导加载程序状态机的实现

引导加载程序状态机执行图3阴影部分的功能。有4种状态：INITIAL_STATE、WAIT_FOR_S、WAIT_FOR_0和WAIT_FOR_EOL。使用XON/XOFF协议，以便可以使用PC上的超级终端来传输s-record文件。

使用名为temp_ptr的临时指针来解析循环缓冲区中收到的数据。当状态机刚刚启动时，以及缓冲区为空时，3个指针data_in、data_out和temp_ptr将指向缓冲区中的相同位置。下图通过流程图显示了引导加载程序状态机的实现。

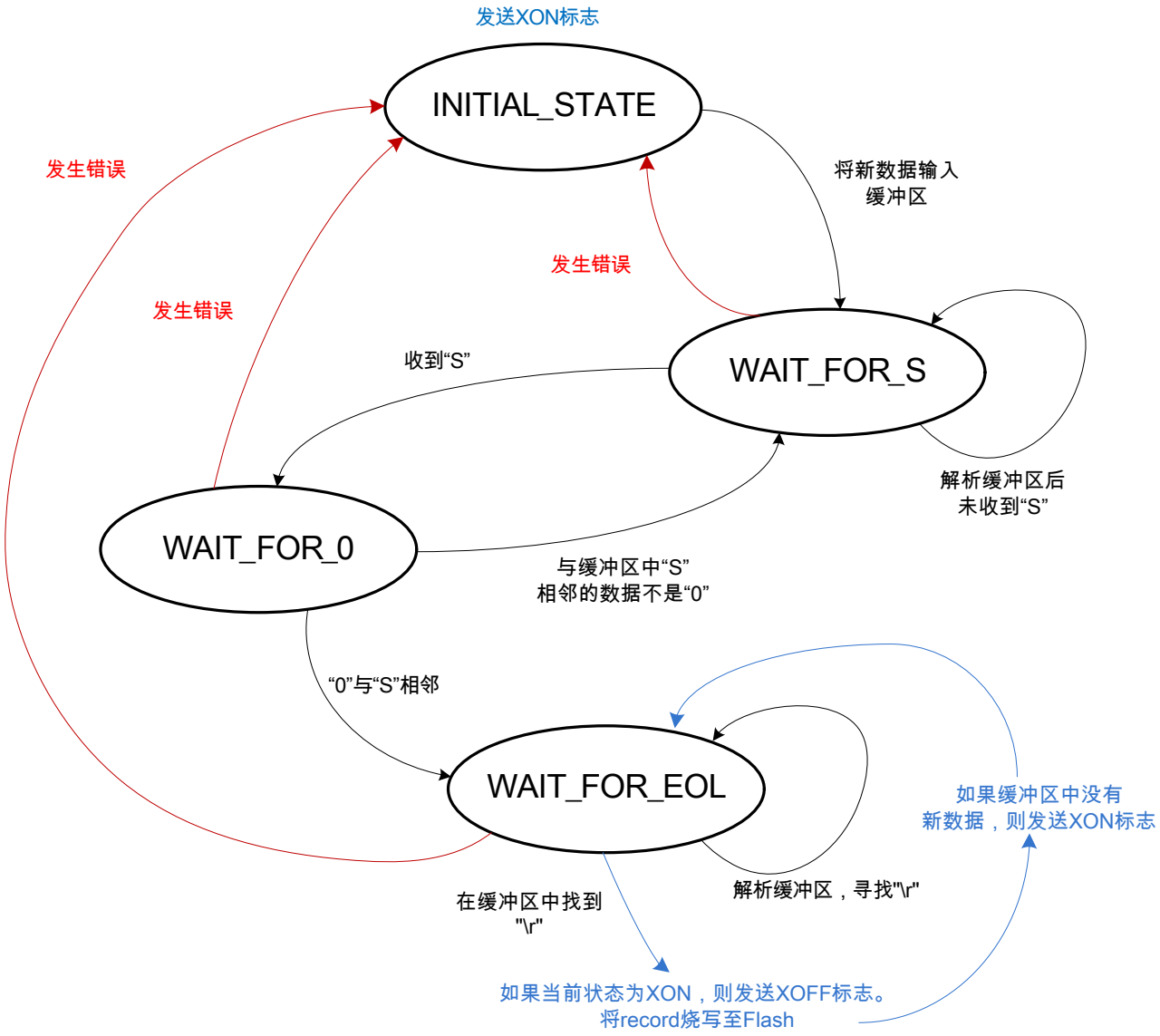

图 8. 状态机流程图

DSC的s-record文件始终以S0记录开头，其后为多个S3记录，结尾为S7记录。因此，如果主机通过SCI传输此文件，则会遵循相同的顺序。下面解释了图8中所示的状态机。

1. 处于INITIAL_STATE状态时，会将XON标志发送到主机，以通知主机开始传输s-record文件。
2. 如果在指定的时间段（大约10秒）后缓冲区仍然保持为空，程序将绕过引导加载程序而跳转至用户应用代码。
3. 如果在指定的时间内更新了缓冲区，则会进入WAIT_FOR_S状态，并停止超时计数器。如果指针data_in不等于temp_ptr，则表示缓冲区中有新的数据输入。

4. 处于WAIT_FOR_S状态时，将通过调用*get_char (&temp_ptr)*来解析缓冲区。如果在缓冲区中找到了“S”，则会进入WAIT_FOR_0状态。否则，将保持为WAIT_FOR_S状态，并解析缓冲区以获取“S”。必须注意，由于通信状态为XON，主机在此状态下将不断地向DSC发送数据。
   - 如果波特率较低，则仅当缓冲区中有新数据时，才解析缓冲区，这样就不会出现问题。
   - 如果波特率太高，则可能会出现问题，因为在调用*get_char(&temp_ptr)*解析旧数据之前，新数据可能与旧数据重叠。在此状态下，指针data_out始终与*temp_ptr*同步，因此，data_out将始终指向“S”驻留的位置。
5. 处于WAIT_FOR_0状态时，如果在缓冲区中发现“0”与“S”相邻，则会进入WAIT_FOR_EOL状态；否则将返回到WAIT_FOR_S状态。
6. WAIT_FOR_EOL状态中，记录中的代码段将烧录至Flash。
   a) 首先，将通过调用*get_char(&temp_ptr)*继续解析缓冲区，直到在缓冲区中发现EOL标志“\r”。
   b) 发现“\r”后，将向主机发送XOFF标志，以通知主机暂停数据传输，同时，记录中由EOL标志指示的代码段将烧写至Flash存储器。由于data_out指向此记录的“S”驻留的位置，因此，可以使用该指针来分析记录，以获取代码段。
   c) 将当前记录烧写至Flash中后，data_out会自然而然地指向下一条记录的“S”驻留的位置，同时，*temp_ptr*将与data_out同步以搜索下一个EOL （即“\r”字符）。

图9和图10显示了每种引导加载程序状态时的循环缓冲区状态。

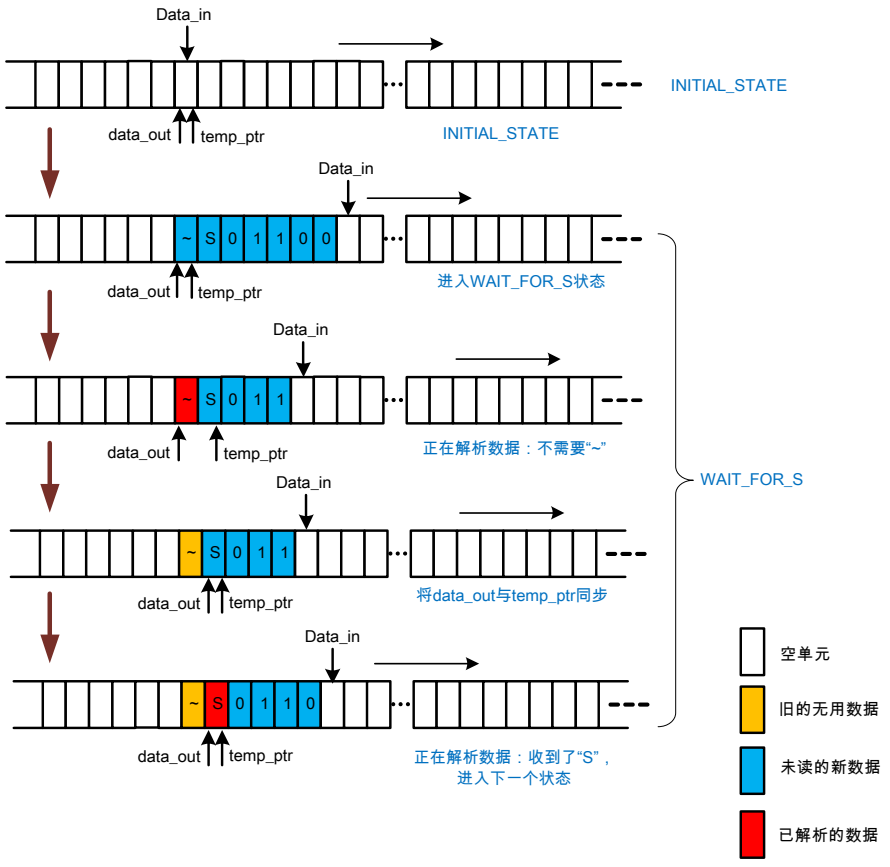

图 9. 缓冲区在 INITIAL_STATE 和 WAIT_FOR_S 状态下的工作方式

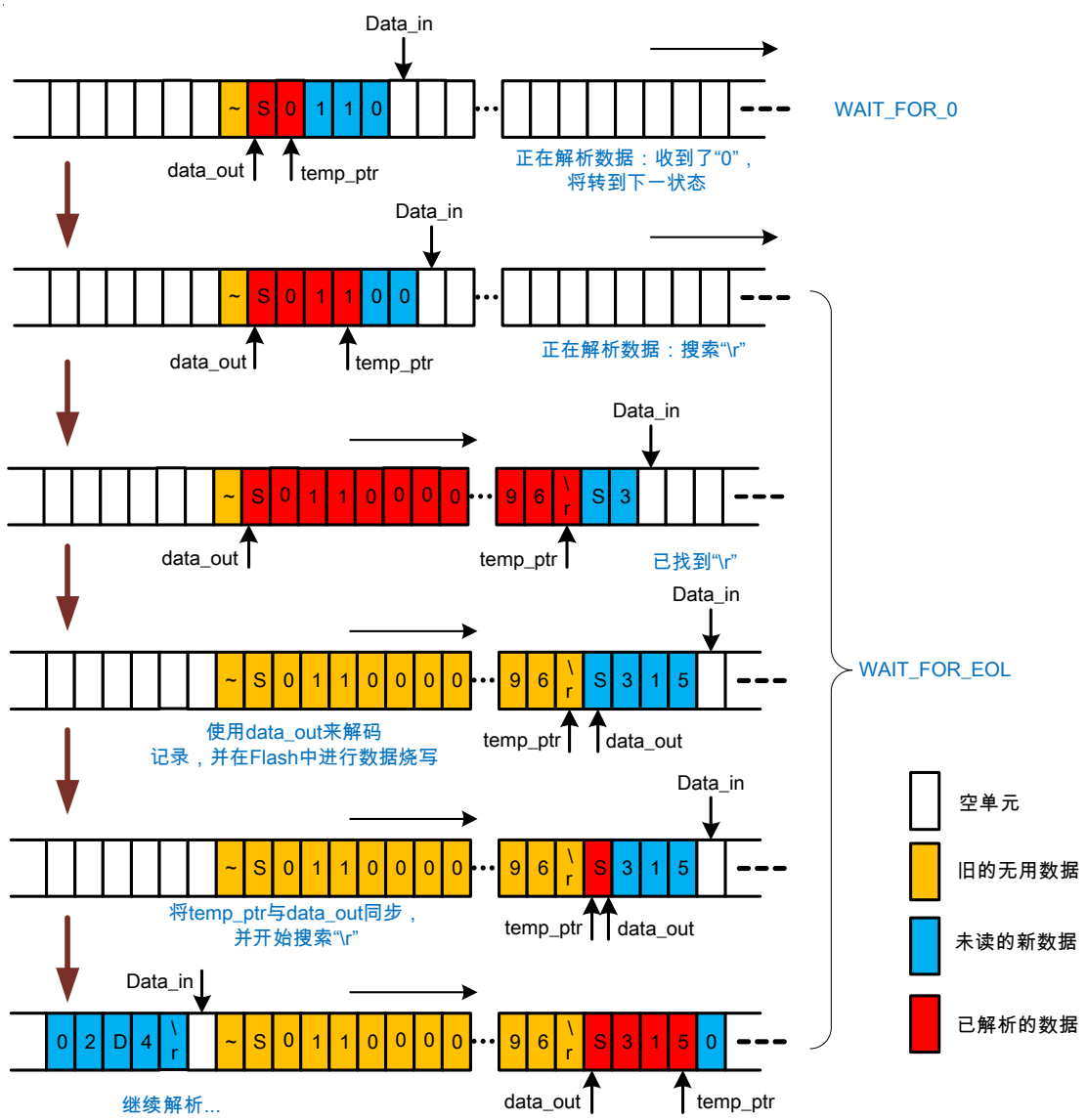

图 10. 缓冲区在 WAIT_FOR_0 和 WAIT_FOR_EOL 状态下的工作方式

收到S7记录后，引导加载程序的执行将会结束，而程序将会根据引导加载程序的存储器配置中所述跳转到用户应用程序。下图显示了实现此状态机的详细流程图。

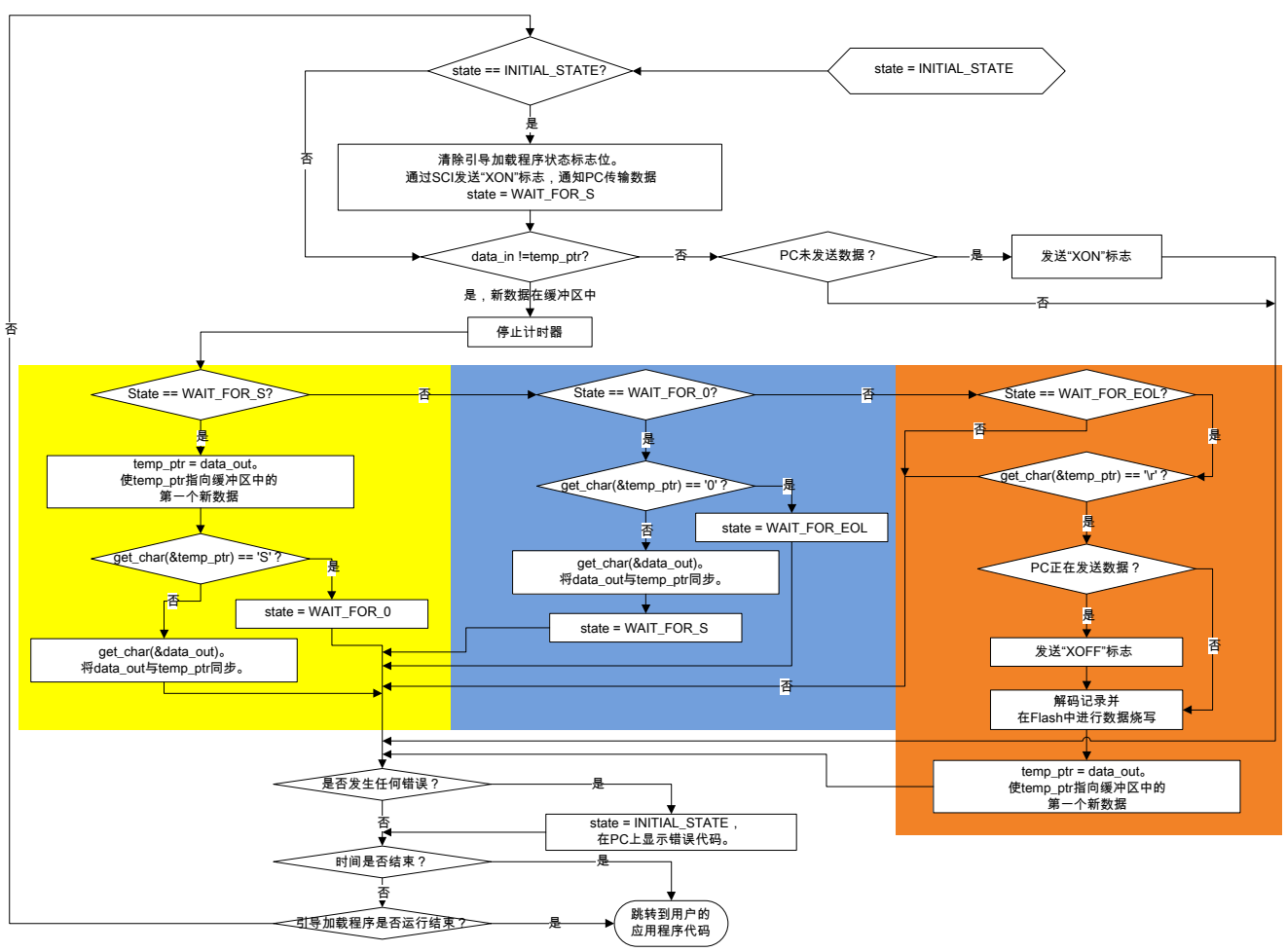

图 11. 引导加载程序状态机的详细流程图

3.4 s-record解码

如引导加载程序状态机的实现中所述，在缓冲区中发现相应的EOL后，将对此记录解码，并将其烧录至Flash。以下部分描述了相关的工作原理。

解码开始后，指针data_out始终指向记录的第一个字符（如图10中所示）。使用函数*srec_decode()*与指针data_out来解码记录。以下是解码一个s-record所要执行的步骤列表。图12显示了这些步骤的流程图。

1. 解码开始时，指针data_out指向记录的开头（即"S"）。
2. 调用*get_char(&data_out)*两次以获取记录类型，此时，data_out指向地址段的第一个字符。
3. 调用*get_byte(&data_out)* 4次以提取地址。每次调用*get_byte()*时，将计算校验和。
4. 根据类型处理记录的代码/数据段。

- 对于类型0 （S0），将通过调用*get_byte(&data_out)*提取代码/数据段，并将其发回到主机，因此，主机将收到字符串“PROGRAM&DATA”。
- 对于类型3 （S3），将提取代码/数据段，并根据地址将其编程到Flash存储器中。
- 对于类型7（S7），将提取代码/数据段，但会忽略它。

5. 处理代码/数据段后，将调用*get_byte(&data_out)*以获取校验和，并将获取的值与计算的值进行比较。通过再次调用*get_byte(&data_out)*将指针data_out指向下一条记录的开头。
6. 处理类型7的记录后，引导加载程序执行结束。

3.5 擦除/编程Flash存储器

只需在FTFL_FCCOBn寄存器中输入适当的值（例如命令代码和数据），即可配置、擦除和编程Flash存储器。通过清除FTFL_FSTAT寄存器的bit 7开始执行设置的命令。有关操作MC56F84xxx的Flash存储器的详细说明，请参见**freescale.com**上的*MC56F847XXRM：MC56F847xx参考手册*。

在执行引导加载程序期间，以下3个命令至关重要（其他有些命令无需使用）。

- 擦除Flash块命令
- 擦除Flash扇区命令
- 编程长字(Longword)命令

如图1中所示，这些命令中使用的Flash地址不是程序存储器映射或数据存储器映射中出现的实际地址。

Flash命令中使用的地址是字节地址，而映射到程序/数据存储器中的地址是字地址对于程序Flash (program flash)，命令地址从0x000000开始；对于数据Flash (data flash)，命令地址从0x800000开始。

例如，一个值为0x1122的16位数据存储在映射到程序存储空间的地址0x0001。从使用Flash命令的角度来看，8位数据0x22存储在字节地址0x0002中，而8位数据0x11存放在字节地址0x0003中。数据Flash适用相同的规则。图13显示了字地址与字节地址之间的关系。

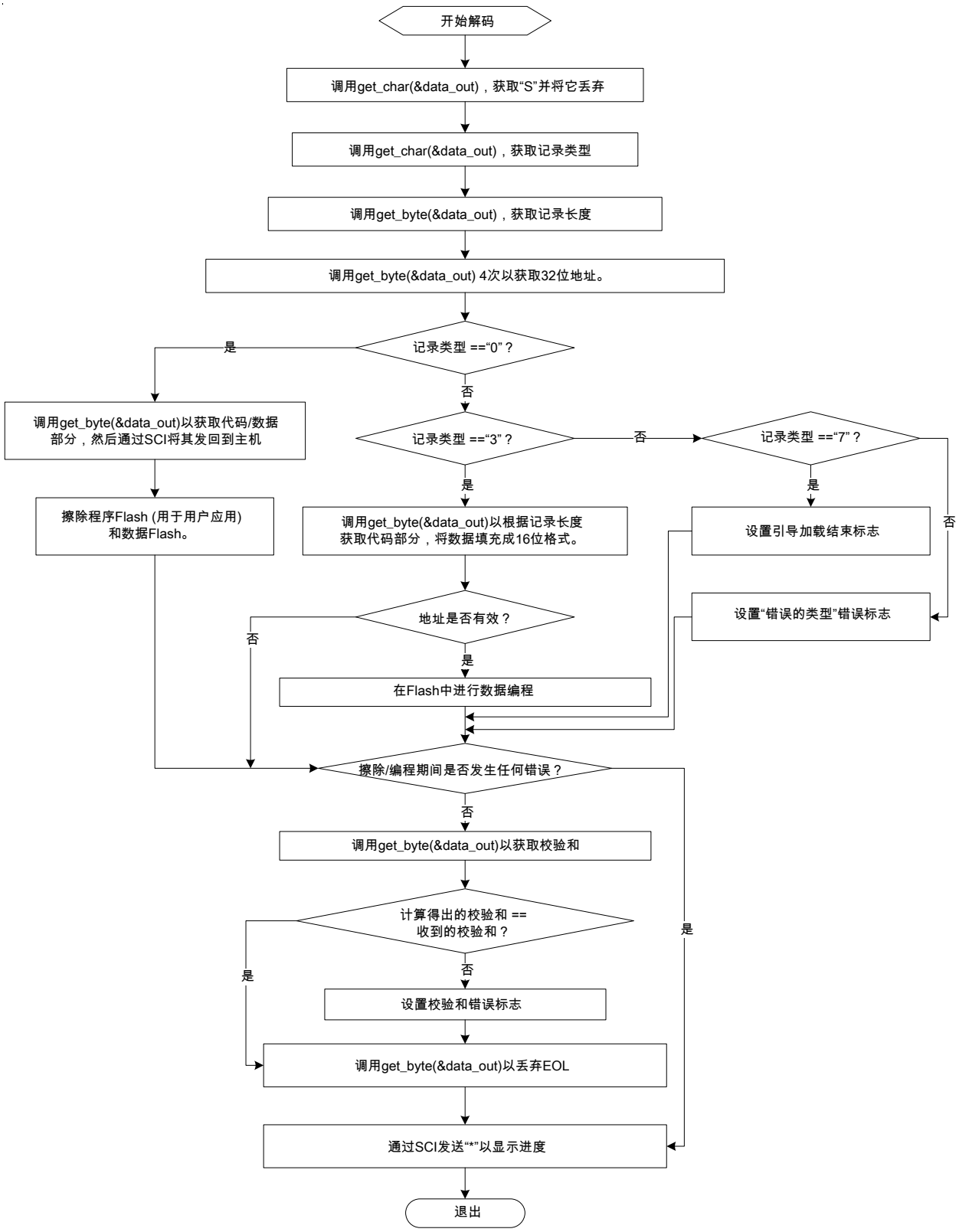
开始解码
调用get_char(&data_out)，获取“S”并将它丢弃
调用get_char(&data_out)，获取记录类型
调用get_byte(&data_out)，获取记录长度
调用get_byte(&data_out) 4次以获取32位地址。
记录类型 ==“0”？
是
否
调用get_byte(&data_out)以获取代码/数据部分，然后通过SCI将其发回到主机
擦除程序Flash (用于用户应用)和数据Flash。
记录类型 ==“3”？
否
记录类型 ==“7”？
是
是
调用get_byte(&data_out)以根据记录长度获取代码部分，将数据填充成16位格式。
设置引导加载结束标志
否
地址是否有效？
设置“错误的类型”错误标志
否
是
在Flash中进行数据编程
擦除/编程期间是否发生任何错误？
否
是
调用get_byte(&data_out)以获取校验和
计算得出的校验和 ==收到的校验和？
是
否
设置校验和错误标志
调用get_byte(&data_out)以丢弃EOL
通过SCI发送“*”以显示进度
退出

图 12. 解码一条记录

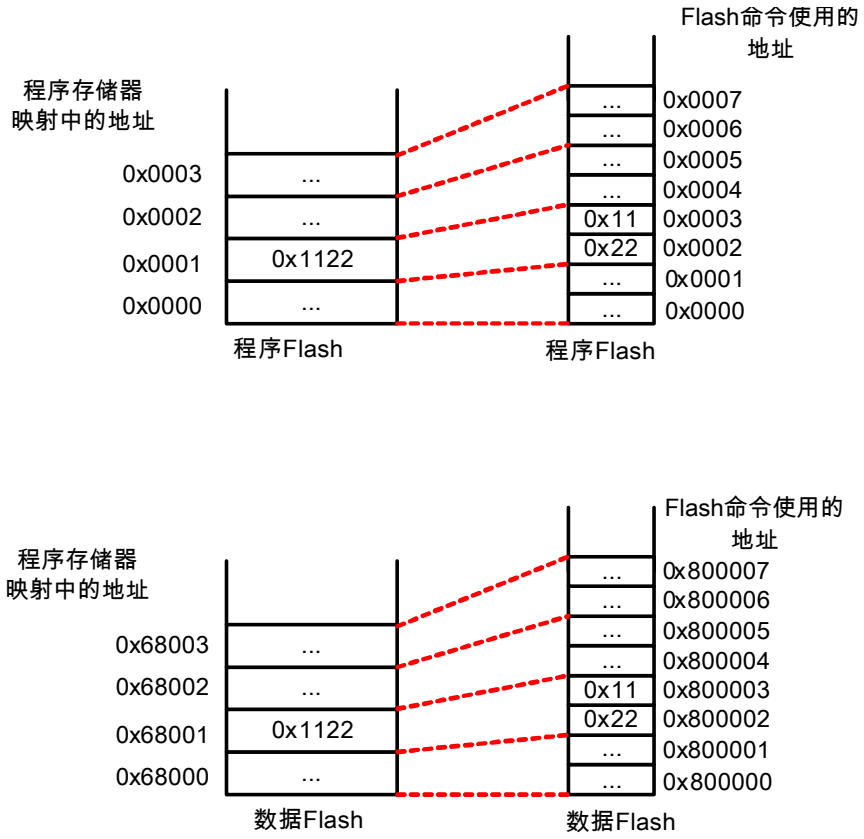

图 13. 字地址与字节地址之间的关系

必须注意，记录中的32位地址使用位25来指示该地址是在程序存储器还是数据存储器中。值1表示数据存储器。

检测到“S3”记录后，将提取代码/数据段，并将其输入到*srec_decode()*函数中的16位数组内。这个16位数组、从记录解码的32位地址以及记录长度将作为参数传递到Flash编程函数*hfm_command()*。此函数首先会分析32位地址，并通过检查其位25来确定它是程序存储器地址还是数据存储器地址，然后，将它更改为Flash命令使用的字节地址。只需使用“Word8 *”类型指针就能将包含代码的16位数组变换为8位数组。编译器将自动处理图13中的关系。

如果处理中的记录为类型0，将会擦除程序Flash和数据Flash。在处理类型3的记录时，将使用编程长字命令。编程长字表示从命令地址开始将4字节数据编程到Flash中，命令地址必须进行长字对齐，也就是说，此地址的最后两位必须是0b00。有时，命令地址没有长字对齐。图14显示了这种情况的处理方式。

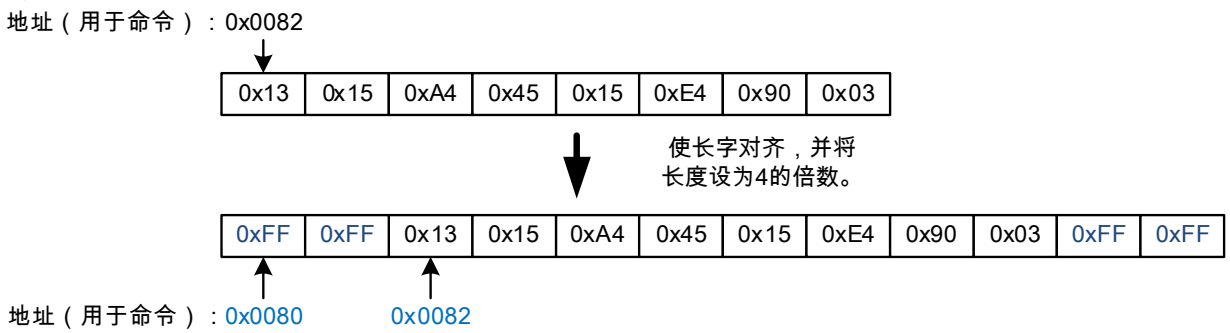

图 14. 在命令地址没有长字对齐时处理字节数组

如图14中所示，起始地址为0x0082，而此地址不是长字对齐。在此数组的前面添加双字节数据0xFF，使起始地址变为长字对齐的0x0080。将0xFF编程到单元中其实没有意义，因为无法通过Flash编程命令将“0”更改为“1”。这里也在该数组的末尾添加了双字节数据0xFF，目的是使该数组的长度成为4字节的倍数，因为编程长字命令每次都是编程4个字节。

4 软件简介

此引导加载程序的代码是在基于PE的CodeWarrior v10.3 （内部版本ID：121211）中开发的。本部分将介绍有关在CW v10.3中开发此代码的某些关键要点。

4.1 链接器文件

PE生成的默认链接器文件不适用于这个特定的应用。必须根据图2修改此文件。MEMORY部分代码显示如下。

```
MEMORY {
    .pIntvectorBoot (RWX): ORIGIN = 0x00000000, LENGTH = 0x000000F0 # Reset and cop vectors
    .pFlashConfig  (RWX): ORIGIN = 0x00000200, LENGTH = 0x00000008 # Reserved for Flash IFR value
    .ppFlash      (RWX): ORIGIN = 0x00000208, LENGTH = 0x0001F1F5 # Primary flash for user code
    .DelayT           (RWX): ORIGIN = 0x0001F3FD, LENGTH = 0x00000003 # Bootloading delay time & user code start position
    .pFlash           (RWX): ORIGIN = 0x0001F400, LENGTH = 0x00000C00 # Primary flash for boot loader, 3Kwords

    .xRAM_bss    (RW) : ORIGIN = 0x00000000, LENGTH = 0x00000800 # 2Kwords for bss
    .xRAM_data   (RWX): ORIGIN = 0x00000800, LENGTH = 0x00000200 # 0.5Kwords for global variables
    .xRAM             (RW) : ORIGIN = 0x00000A00, LENGTH = 0x00001600 # 5.5Kwords for heaps and stacks
        .pRAM_code  (RWX): ORIGIN = 0x00062000, LENGTH = 0x00002000 # 8Kwords for code
        .xRAM_code  (RW) : ORIGIN = 0x00002000, LENGTH = 0x00002000 # mirror of .pRAM_code
}
```

建议在映射到程序空间的RAM中执行Flash擦除/编程函数*hfm_command()*（参见擦除/编程Flash存储器）。因此，该函数必须存储在Flash存储器中，但在RAM中运行，这可以根据以下SECTIONS部分的代码描述来实现。

```
.ApplicationCode :
{
        _pFlash_code_start  = .;
        # Note: The function _EntryPoint should be placed at the beginning of the code
        OBJECT (F_EntryPoint, Cpu_c.obj)
        # Remaining .text sections
        * (rtlib.text)
        * (startup.text)
        * (fp_engine.text)
        * (user.text)
        * (.text)
        # save address for the data starting in pROM
        Fpflash_mirror = .;
        Fpflash_index  = .;
} > .pFlash

.prog_in_p_flash_ROM : AT(Fpflash_mirror)
 {
        Fpram_start = .;
        _pram_start = .;

        * (interrupt_vectors.text)
        * (pram_code.text)

        # save data end and calculate data block size
        Fpram_end = .;
        Fpram_size = Fpram_end - Fpram_start;
        _pram_size = Fpram_size;
        Fpflash_mirror2 = Fpflash_mirror + Fpram_size;
        Fpflash_index = Fpflash_mirror + Fpram_size;
} > .pRAM_code
```

在链接器文件中定义全局常量，以识别此函数的大小及其在Flash和RAM中的起始地址。

```
F_pflash_code_start = Fpflash_mirror;                   # start address in Flash
F_dram_code_start  = _pram_start - 0x60000;    # start address in RAM (the address mapped into data memory)
F_dram_code_size  = _pram_size;            # size
```

一进入*main()*函数，就会使用*mem_copy()*函数将此函数代码从Flash存储器复制到RAM。未使用PE生成的“56F83x_init.asm”中的默认存储器复制例程，因此，必须根据以下代码中所示，在链接器文件中将两个常量设置为0。

```
F_xROM_to_xRAM = 0x0000;
F_pROM_to_xRAM = 0x0000;
```

mem_copy 函数如以下代码所示。

```
asm void mem_copy(long p_start,long x_start,unsigned int cnt)
{
        move.l  a10,r2
        move.l  b10,r3
        do      y0,>>end_prom2xram      // copy for 'cnt' times
        move.w  p:(r2)+,x0              // fetch value at p-address
        nop
        nop
        nop
        move.w  x0,x:(r3)+              // stash value at x-address
end_prom2xram:
        nop
        rts
}
```

PE生成的默认配置为小数据模型（Small Data Model，SDM），这意味着只能使用16位指针，该指针可在一定程度上加速引导加载程序的执行。如图2所示，代码从17位地址0x1F400开始；为了避免编译错误，按以下代码行中所示，将函数地址*F_pflash_code_start*和*F_dram_code_start*分割成两个部分。

```
F_pflash_code_start_h = (F_pflash_code_start/65536)&0xffff;
F_pflash_code_start_l = F_pflash_code_start &0xffff;
F_dram_code_start_h = (F_dram_code_start/65536) &0xffff;
F_dram_code_start_l = F_dram_code_start &0xffff;
```

使用以下代码调用函数*mem_copy*。

```
mem_copy(((Word32)(&_pflash_code_start_h)<<16) + (Word32)(&_pflash_code_start_l),\
        ((Word32)(&_dram_code_start_h)<<16) + (Word32)(&_dram_code_start_l),(UWord16)&_dram_code_size);
```

以类似的方式处理.data段。PE生成链接器文件后，每次编译项目时，用户可以通过在“Generate linker file”选项中选择“no”来禁止PE生成链接器文件。下图显示了可以在哪个位置更改此设置。

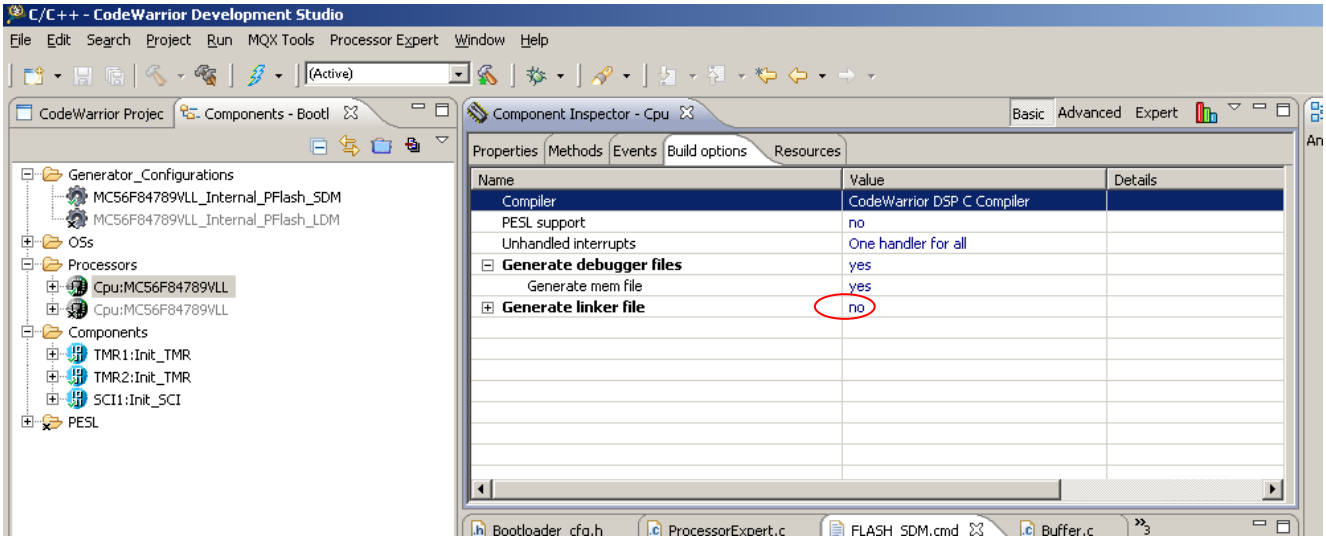

图 15. 用于禁止 PE 生成链接器文件的 PE 选项

有关代码实现的详细信息，请参见本应用笔记随附的源代码项目AN4759SW。

4.2 有关实现引导加载程序的宜忌事项

不熟悉CodeWarrior v10.3和MC56F84xxx DSC系列器件的用户需了解以下要点。

- 默认情况下已启用看门狗。
- 引导加载程序的起始地址必须正好在扇区的边界上，程序Flash中一个扇区的大小为1K字。这是因为，擦除Flash时是以扇区为单位进行的，也就是说，如果您想要擦除Flash存储器中的某些单元，则必须擦除这些单元所属的整个1 K字存储器。
- 使用SCI进行通信并在PE中使用Init_SCI bean时，请记得禁用其DMA功能，否则将会出现接收错误，除非确实使用了DMA功能。
- 更改链接器文件后必须执行“Clean project”，否则该文件将不起作用。
- 本应用中使用的循环缓冲区是通过内核的模寻址函数实现的，它要求起始地址是偶数值。
- 本应用中的SCI中断是快速中断，也就是说，程序将直接转到中断服务例程，而不会先跳转到向量表。因此，如果不使用快速中断，则应更改中断向量表的起始地址，以避免用户应用程序与引导加载程序之间发生冲突。

5 用户应用程序要求

如果用户应用程序基于PE开发，则必须考虑以下两个问题，以便能够通过上面开发的引导加载程序正常下载该应用程序，并且引导加载程序仍然能够与用户应用程序配合工作：

- 应该禁止PE生成的链接器文件自动更新。图15显示了设置方法。图2所示的程序Flash中用于代码的存储范围不能超出0x1F3FF。必须在地址范围0x1F3FD-0x1F3FF内输入用户应用程序的起始地址和延迟时间。

```
MEMORY {
……
.p_Code  (RWX) : ORIGIN = 0x00000208, LENGTH = 0x000F000
……
        .xBootCfg (RWX): ORIGIN = 0x1F3FD, LENGTH = 3
}



SECTIONS{
        .ApplicationConfiguration:
        {
                # Store the application entry point
                WRITEW(F_EntryPoint);  # write 4 bytes

                # Boot loader start delay in seconds
                WRITEH(12);    # write 2 bytes
         } > .xBootCfg
}
```

- 应该对PE生成的“Vector.c”做出修改。必须按以下代码中所示更改*_vect()*函数中的前两个指令，以便在复位芯片后，程序首先会跳转到引导加载程序。

```
#define boot_start 0x1f400

#pragma define_section interrupt_vectors "interrupt_vectors.text"  RX
#pragma section interrupt_vectors begin
volatile asm void _vect(void) {
JMP  boot_start              /* Interrupt no. 0 (Used)   - ivINT_HW_RESET */
JMP  boot_start              /* Interrupt no. 1 (Used)   - ivINT_COP_RESET  */
                ……
                ……
}
#pragma section interrupt_vectors end
```

根据上面的内容更改 Vector.c 后，此文件就不能再改动了。下图显示了如何冻结 PE 的代码生成功能。

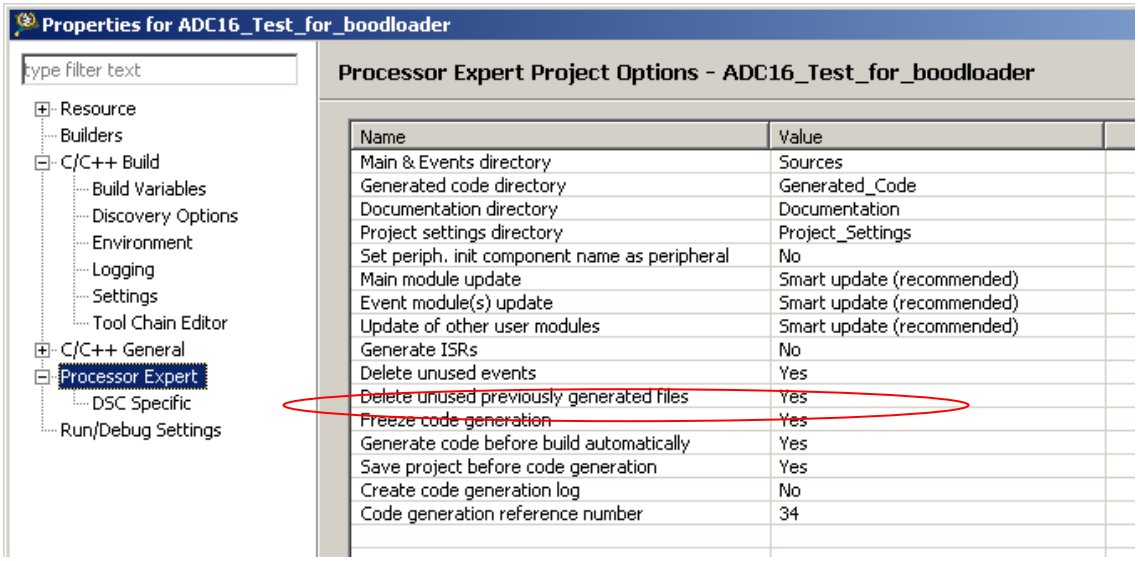

图 16. 用于冻结 PE 代码生成功能的选项

1. 默认情况下不会生成s-record；应该按图17中所示启用生成s-record。必须选择“Sort by Address”选项，并且s-record最大长度（“Max S-Record Length”）不能超过255。S-record EOL字符（“S_Record EOL Character”）应该为DOS（\\r\\n）。
2. 有关如何使用超级终端和此引导加载程序来实现引导加载程序功能的信息，请参见 **freescale.com**上的 *AN4275：面向56F82xx的串行引导加载程序*。

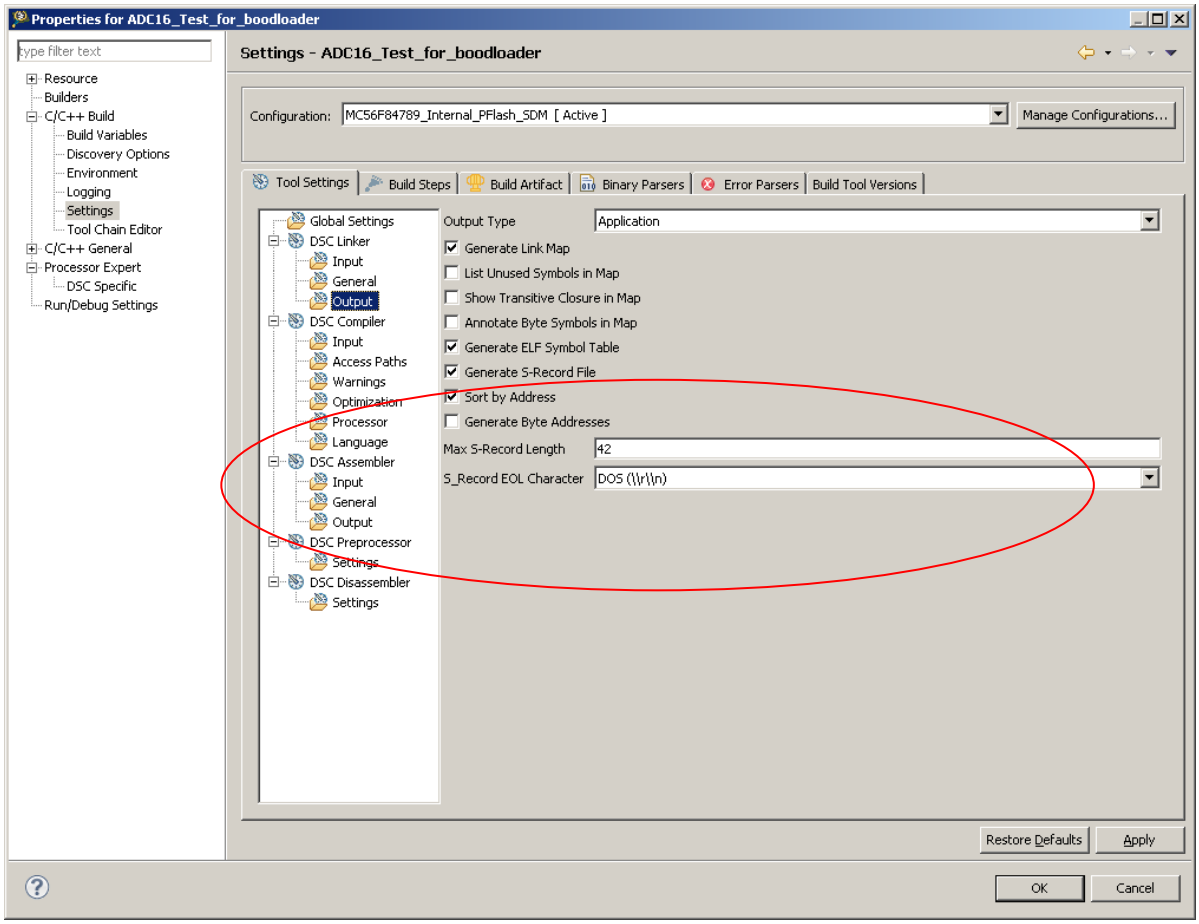

图 17. 用于生成 s-record 的配置

6 结语

本应用笔记提供了在56F84xxx系列器件上实施引导加载程序的方法。本文档还详细介绍了引导加载程序的工作原理，以及如何使用装有Processer Expert的CodeWarrior10.3实施引导加载程序。MC56F84xxx和MC56F827/3xx DSC系列器件都可以使用示例代码作为参考。

7 参考

- **freescale.com**中提供的*MC56F847XXRM：MC56F847xx参考手册*
- **freescale.com**中提供的*AN4689：MC56F84xxx DSC上的EEPROM*
- **freescale.com**中提供的*DSP56800E和DSP56800EX参考手册*
- **freescale.com**中提供的*AN4275：面向56F82xx的串行引导加载程序*

8 修订历史记录

修订版号	日期	重大变更
0	06/2013	初始版本

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飞思卡尔产品。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。飞思卡尔保留对此处任何产品进行更改的权利，恕不另行通知。

飞思卡尔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局限于后果性的或附带性的损害在内的所有责任。飞思卡尔的数据手册和/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。所有操作参数，包括“典型值”在内，必须由客户的技术专家根据每个客户应用进行验证。飞思卡尔未转让与其专利权及其他权利相关的许可。 飞思卡尔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, CodeWarrior, and Processor Expert 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.

© 2013 Freescale Semiconductor, Inc.

Document Number: AN4759
Rev. 0, 06/2013
June 23, 2013