Freescale Semiconductor
应用笔记

Document Number: AN4723
Rev 0, 07/2013

S12Z MagniV 引导加载程序

作者: Arturo Inzunza

内容

1 简介

本应用说明介绍了 S12Z 内核引导区的工作原理，以及如何实现和使用引导加载应用程序对器件进行串行重新编程。S12Z 内核是 MagniV 系列与其他飞思卡尔 16 位微控制器的主要区别之一。本文档重点描述该内核是如何管理引导进程、存储器布局和操作机制，以及中断管理。本文档提供的引导加载应用程序可与任何串行通信协议驱动程序相匹配，为实现提供灵活性。一些串行引导加载程序的常用备选模块包括 SCI、CAN 或 LIN，但也可以使用 I2C、SPI 或常规通用 IO（Bit-Banging）。引导加载应用程序与通信驱动程序明确分离，以便于在不同应用（和/或 MagniV 器件）间替换和移植引导加载程序。

S12Z MagniV 引导加载程序旨在提供交叉平台解决方案，可方便地移植到不同的 MagniV 器件。它利用所有具有 S12Z 内核的 MagniV 器件共享类似架构这一优势。引导加载程序设计得很小，从而可以用于更小型器件，而无需消耗大部分的存储资源。以下章节将重点描述 S12Z 内核的操作特性、S19 记录格式、开发被引导加载程序代码时的注意事项，以及对 PC 应用程序服务器的快速说明。PC 应用程序可通过串行 RS-232 端口传输 S19 文件到目标。

2 S12Z 内核工作原理

S12Z 内核与飞思卡尔其他 16 位内核相比存在许多区别。最重要的是采用了 24 位寻址机制。早期的 16 位内核具有 16 位寻址，仅支持 64 KB 的直接存储器寻址。此限制强制

使用页来实现更大的存储器，而对于这些存储器的控制也变得更为复杂。S12Z 内核的 24 位寻址能力使其最多可以进行 16 MB 的直接存储器寻址。这意味着不再需要页，不过地址长度为 3 字节长。S12Z 内核还具有一个经细微修改的引导机制，比早期内核上的更为简单。

2.1 存储器布局

S12Z MagniV 器件家族采用多种不同的存储器尺寸，但都共用一种常用布局。利用更大寻址能力这一优势，所有存储器：RAM、程序 Flash（P-Flash）和 EEPROM 均可被直接寻址，而无需使用页。本文不会对 EEPROM 进行分析，因为引导加载程序不会对此存储器进行擦除、编程或读取操作。

P-Flash 存储器尺寸范围从 128 KB（较大器件）到 8 KB（较小器件）。不管 P-Flash 存储器有多大，它总是位于寻址空间的最后，并在地址 0xFFFFFF 处结束。下图显示了 32-KB 存储器的 P-Flash 分布和存储器保护选项示例。

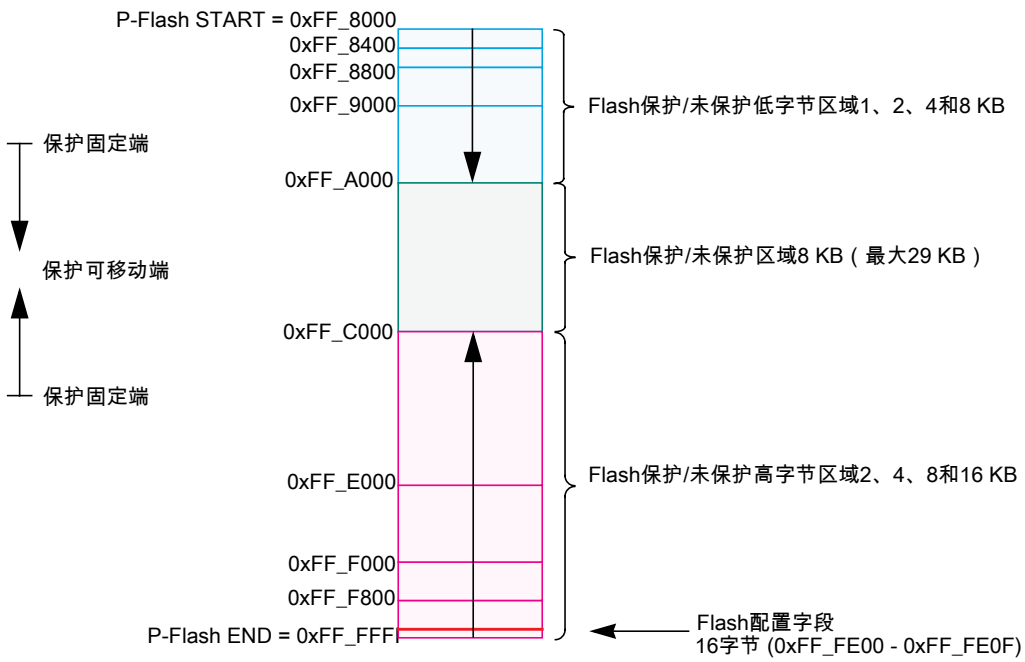

图 1. 32-KB P-Flash 布局

MagniV 器件上使用的存储器技术由分成多个 512 字节扇区的单个区块组成。扇区是存储器控制器可擦除的最小单元。下表显示了 MagniV 器件上不同存储器大小的 P-Flash 存储器布局数据。

表 1. P-Flash 存储器布局信息

内存大小	起始地址	结束地址	扇区数量
128 KB	0xFE0000	0xFFFFFF	256
64 KB	0xFF0000	0xFFFFFF	128
32 KB	0xFF8000	0xFFFFFF	64
16 KB	0xFFC000	0xFFFFFF	32
8 KB	0xFFE000	0xFFFFFF	16

P-Flash 存储器的最后一个扇区中存放了向量表和 NVM 控制器的配置信息。该扇区及其详细信息将在本文档的后续章节中进行分析。除了 P-Flash 存储器，RAM 存储器也包括在通用寻址映射中。

RAM 大小与 P-Flash 大小一样随器件而异。最大的 MagniV 器件具有 8 KB RAM, 而最小的仅为 1 KB。MagniV 器件上的 RAM 存储器总是从地址 0x1000 开始。下表显示了不同设置下的 RAM 存储器布局。

表 2. RAM 存储器布局信息

内存大小	起始地址	结束地址
4 KB	0x1000	0x2000
2 KB	0x1000	0x1800
1 KB	0x1000	0x1400

引导加载程序利用 Flash 和 RAM 的起始和结束地址信息来确定擦除哪些存储器扇区，以及哪些 RAM 区域用于存储数据。

2.2 中断向量和 NVM 配置

P-Flash 的最后一个扇区用于存储中断向量信息和 P-Flash 保护配置。该 NVM 部分上的保护字节在上电复位(POR)时被复制到存储器控制器寄存器中。表 3 显示了位于 P-Flash 最后一个扇区中的保护字节的地址和名字。

复位时，最后一个 P-Flash 扇区上的 FPROT、DFPROT、FOPT 和 FSEC 字节将复制到存储器控制器寄存器空间上具有相同名字的寄存器中。实际的存储器保护功能取决于存储器控制器上的寄存器，因而运行中的应用程序可在运行时修改保护配置。在下次复位时，将再次从 P-Flash 把默认值复制到寄存器中。在 P-Flash 上修改保护值将确保在每次复位后，都会默认加载新的配置

表 3. P-Flash 配置字节

全局地址	大小（字节）	说明
0xFFFE00-0xFFFE07	8	后门比较密钥
0xFFFE08-0xFFFE09	2	保护覆盖比较密钥
0xFFFE0A-0xFFFE0B	2	保留
0xFFFE0C	1	P-Flash 保护字节(FPROT)
0xFFFE0D	1	EEPROM 保护字节(DFPROT)
0xFFFE0E	1	Flash 非易失性字节(FOPT)
0xFFFE0F	1	Flash 加锁寄存器(FSEC)

除了 P-Flash 保护字节，最后一个 P-Flash 扇区中还保存了默认的中断向量表，地址为 0xFFFE10 至 0xFFFFFF。中断向量表中保存了每个必须用于服务中断请求的函数的指针。虽然函数指针为 24 位长，但为了兼容，中断函数指针存储为 32 位地址（4 字节）。S12Z 内核可以寻址最多 124 个中断源，不过不会实现所有。中断源的可用性取决于每个特定 MagniV 器件的设置。有关特定中断详细信息，请查看具体 MagniV 微控制器参考手册中的“中断向量位置”表。可通过修改 IVBR 寄存器将中断表从默认位置移开。

虽然每个 MagniV 微控制器具有不同的可用中断, 但所有这些中断具有相同的启动向量。中断表的最后一个中断向量保留为该启动向量。POR 后，MCU 将访问最后一个向量，位于最后一个 P-Flash 扇区（0xFFFFFC-0xFFFFFF [4 字节]），并将自动跳转到该指针。即使中断表可以移动，复位后中断表会回到其默认位置（最后一个 P-Flash 扇区），因此启动向量总是为 P-Flash 阵列的最后 4 个字节。在早期的 16 位内核上，复位向量根据复位原因被放置在不同位置，看门狗复位跳转到的位置会与外部引脚复位不同。无论何种复位原因，S12Z 只有一个复位向量。

2.3 存储器操作

MagniV 器件上的存储器控制器称为 FMTRZ，可在运行时擦除和写入非易失性存储器扇区，而无需额外供电。FTMRZ 模块的一些操作特性包括：

- 最小擦除单元为一个扇区（512 字节）
- 必须先擦除扇区方可写入
- 最小写入大小为 8 字节
- 写操作必须与 8 字节边界对齐
- 不能同时读取和写入 Flash

擦除一个扇区会将所有位置 1。擦除扇区后，FTMRZ 可以尝试对其写入数据。但是在从 Flash 执行代码时不能进行写操作，因为 Flash 不能同时读写。要写入 Flash 扇区，应用程序必须从 RAM 执行代码。需要对写边界特别注意。写操作的地址必须总是与 8 字节边界对齐，这意味着最后 3 位必须为 0，如 0xFFFE08。这是因为最小写入大小为 8 字节，在本示例中写操作会修改字节 0xFFFE08 至 0xFFFE0F，下一个写操作将发生在 0xFFFE10 上，再次与 8 字节边界对齐。

3 S-Record 格式

引导加载应用程序使用 S-Record 格式的编程文件，称为 S19、SREC 或 Sx。S-Record 格式由 Motorola 开发，用于表示微控制器的存储器内容。有多款编译器可以在编译后直接生成 S-Record 文件，而无需额外工具。S-Record 是推荐的程序表示，因为工具可以方便地解析该简单的文本文件，并对微控制器进行编程。

S-Record 文件按行排列，均以字母“S”开头。每行的头两个字符定义了该行的用途，例如：

- 标识
  - S0：可包含项目名称和供应商信息，由编译器决定
- 数据序列
  - S1：具有 2 字节寻址的实际存储器数据（16 位）
  - S2：具有 3 字节寻址的实际存储器数据（24 位）
  - S3：具有 4 字节寻址的实际存储器数据（32 位）
- 行数
  - S5：文件中数据序列行的数量（S1、S2 或 S3）
- 程序起始地址
  - S7、S8 或 S9。不过通常不使用

S-Record 文件可以只有这些行类型中的一种，例如在一个文件中可以没有 S1 和 S2 记录。由于 MagniV 器件具有 24 位地址，S-Record 文件将用 S2 行表示要发送的数据。

每条记录行由以下元素组成：

- 标识符：行的头两个字符（S0、S2 或 S9 等）。
- 行字节大小：接下来两个字节表示除标识符及其本身以外的行的字节数（十六进制）。
- 地址：接下来的 2、3 或 4 字节表示数据地址，由记录类型决定。
- 校验和：行的最后一个字节。计算除标识符外所有字节的校验和。

下图显示了 MagniV 微控制器的 S-Record 文件示例

```
1  S0580000433A5C53564E5C47726F75705C41727475726F5C4B6E6EF785C536F6674776172655…
2  S224FF81E0A407850A6105ABFE00111E1B901CE7F70B867D0500000010001100FFFF0100012A
3  S213FF820000000000000000001030002C000000000A5
4  S207FFFFFDFF80007E
5  S90380007C
```

图 2. S-Record 文件示例

在上图中，行 1 表示文件的标识和说明（根据 S0 标识符），行 5 指示启动地址（S9 标识符）。引导加载应用程序的运行无需这些记录。行 2、3 和 4 为 S2 记录，表示必须写入存储器的数据，对引导加载应用程序很重要。下表将这些行（S2）拆分为各个元素。

表 4. S2 记录上的数据字段示例

行	标识符	字节数[十六进制]	地址[十六进制]	数据[十六进制]	校验和[十六进制]
2	S2	24	FF81E0	A407850A6105AB FE00111E1B901C E7F70B867D05000 00010001100FFFF 010001	2A
3	S2	13	FF8200	000000000000010 30002C000000000	A5
4	S2	07	FFFFFD	FF8000	7E

S2 行最多为 37 字节长，标识符除外（1 个大小字节 + 3 个地址字节 + 32 个数据字节 + 1 个校验和字节）。S2 行是唯一与 MagniV 引导加载应用程序有关联的 S-Record 行。

4 引导加载程序软件

此应用说明随附的引导加载程序软件设计用于接收 S-Record 程序，并在运行时将其加载到 P-Flash 上。S-Record 程序必须一次输出一行。引导加载程序会对每行数据进行处理， 然后回复应答消息以指示可以发送新一行数据。引导加载程序软件在按顺序接收下一个 S-Record 行的同时，必须管理 P-Flash 的擦/写操作。

4.1 通用架构

引导加载应用程序的设计独立于用于接收 S-Record 数据的通信协议。可修改通信驱动程序以适应客户的应用程序，而引导加载程序软件保持不变。该架构为在不同应用和器件上实现引导加载程序提供了灵活性和简便性。

引导加载应用程序仅需依靠 3 个 MCU 外设模块：时钟和电源管理单元（CPMU）、存储器控制器（FTMRZ）和通用 IO 模块。通信驱动程序独立于引导加载程序。下图显示了这些不同模块之间的联系。

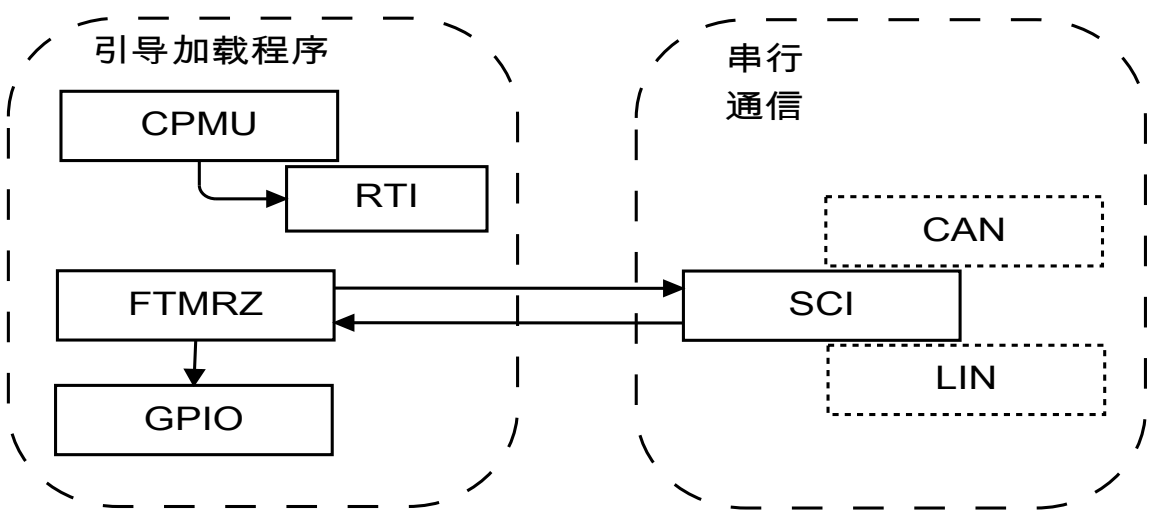

图 3. 引导加载应用程序关系图

复位时，微控制器开始运行引导加载应用程序，并首先配置 CPMU 和 RTI 模块。CPMU 模块配置 PLL，将 BusClock 设为 32 MHz。使用 RTI 来跟踪时间。在可配置的超时后，引导加载程序停止等待更新程序传入，并跳转到主应用程序。

FTMRZ 模块是存储器控制器，负责处理接收到的串行数据并将其写入 P-Flash。该模块还用于擦除所有引导加载程序未占用的 P-Flash 扇区，包括中断向量部分。通用 IO 仅用来配置单个“活动”LED，每当接收到新 S-Record 行时闪烁。这只是活动的视觉指示，可以关闭。

应用程序的串行通信部分与引导加载程序的其余部分无关。可从任何通信协议接收 S-Record，然后将其发送至 FTMRZ 模块进行处理。必须以指定格式存放数据，这样 FTMRZ 才能进行处理。通信驱动程序负责确保数据存放符合相应格式。将在下一章节详细解释该格式。

引导加载应用程序设计得很小，因而无需使用中断。由于需要擦除默认中断向量，引导加载应用程序的中断向量将会需要重定位，这会影响程序大小，因为需要为其分配更多空间。通过轮询通信驱动程序来确保数据接收。下图显示了完整的引导加载应用程序的程序流程图。

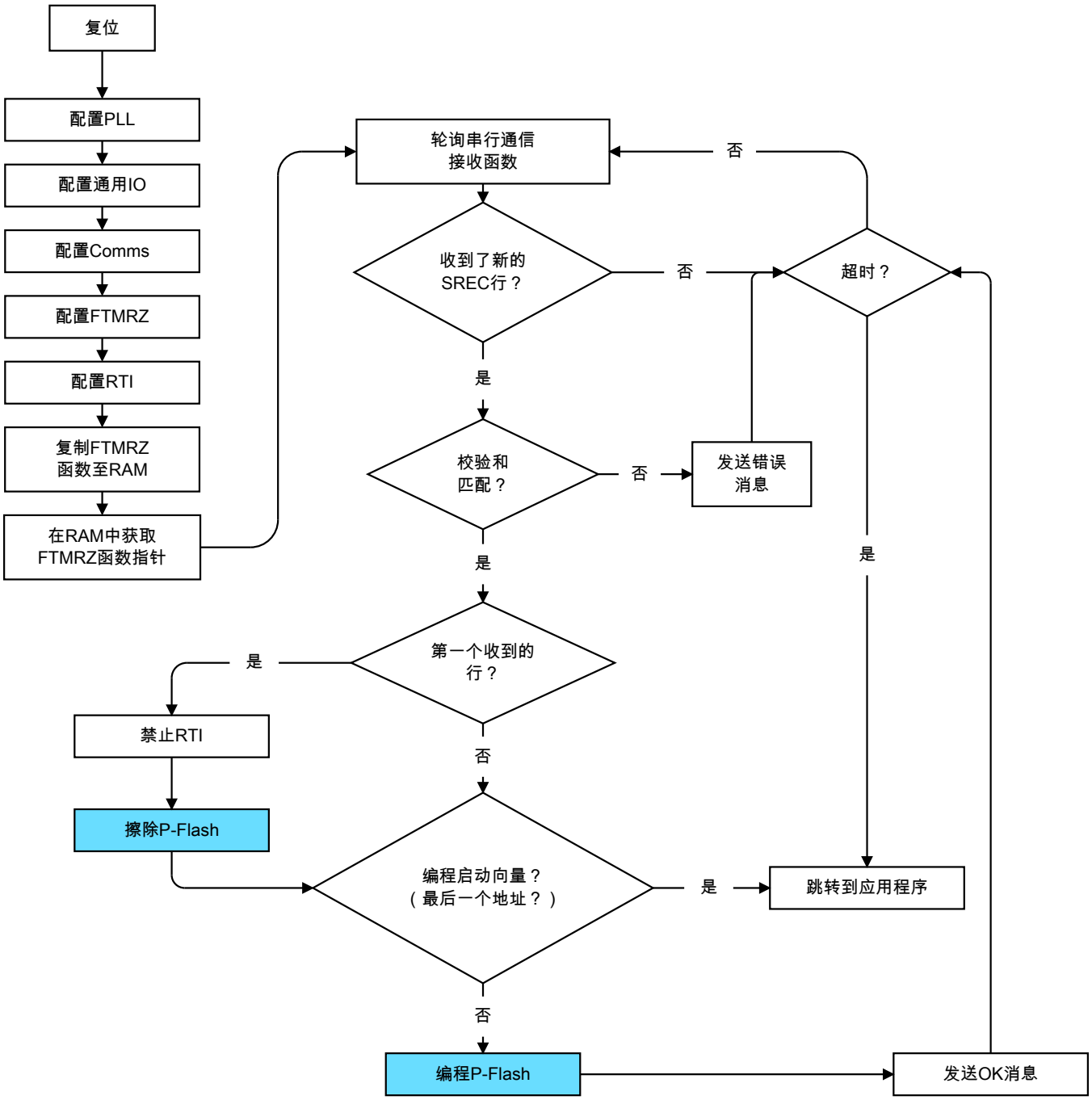

图 4. 引导加载应用程序流程图

引导加载应用程序等待要接收的数据。如果数据在超时前被接收，FTMRZ 模块会擦除 P-Flash 并写入数据。在接收到第一个 S-Record 行后，RTI 模块将被禁止，不会发生超时。这样确保在编程操作期间不会发生超时。当 FTMRZ 需要写入启动向量（P-Flash 的最后 4 字节），将认为编程已完成且不再接收任何数据。

注

已接收程序（S-Record）的启动向量将不会写入 P-Flash 的最后地址。该地址保留用于引导加载程序的启动向量，使引导加载程序在下次 POR 时启动。

当发生超时或已接收程序达到最后地址（且已完成）后，应用程序会跳转到主应用程序。主应用程序必须从固定地址（可在引导加载程序中配置）开始。该地址在编译时即被确定，一旦引导加载程序被加载到目标器件后便不能修改。

在图 4 中，“擦除 P-Flash”和“编程 P-Flash”用蓝色标出，以指示它们从 RAM 执行。” Flash 存储器不能同时读和写，因此这两个修改 Flash 的操作需要从 RAM 执行。在程序启动时，这些函数会复制到 RAM 中，并生成新的 RAM 指针。在擦除或写操作完成后，会返回 Flash 执行，直到需要对新的 S2 记录进行编程。

4.2 数据接收

如上所述，引导加载程序的数据接收部分独立于应用程序的其他部分。任何通信协议可用来接收新程序，用户也可以灵活地采用自己（或第三方）的驱动程序。从引导加载程序调用通信驱动程序来执行不同的任务；以下是必须采用的函数：

- ***void Comms_Init(void)*** — 通信初始化例程。该函数必须用来初始化外设和配置时钟，并根据需要设置通用 IO 配置。该函数在配置 PLL（为 32 MHz）后调用，且只能调用一次。
- ***void Comms_Tx_Ack(enum ACK_OPTS ack)*** — 通信发送应答例程。调用该函数向发送器件发出应答消息。应答消息仅 1 字节长。枚举 ACK_OPTS 只有两种值“OK”或“CRC_Error”。
- ***void Comms_Rx_Data(void)*** — 通信接收数据例程。持续轮询该函数以触发数据接收机制。接收到的数据必须存放在“BootPhraseStructure BP”中，最多可保存 37 个字节（最大 S2 行大小）。
- ***void Comms_Reset_Reg(void)*** — 通信复位寄存器例程。当引导加载程序准备跳转到主应用程序时调用该函数。该函数必须确保由驱动程序修改的所有寄存器均返回为其默认值。这确保了主应用程序开始运行时，MCU 的状态已知，如同从复位启动。

引导加载程序不能使用中断，因此需要轮询 Comms_Rx_Data 函数。每次调用时，该函数必须检查是否接收了新数据，如果是就将其存放于 BootPhraseStructure。该结构的位域与标准 S2 S-Record 行相匹配。该结构大小为 37 个字节，是 S2 记录的最大尺寸 (S2 标识符除外)。BootPhraseStructure 架构允许通过数组或通过具有位域定义的结构来访问各个字节。通信驱动程序应以数组形式写入信息，因为在数据接收阶段无需对信息进行解析。BootPhraseStructure 的框架如下：

```
typedef union
{
     UINT8 Byte[37];     /* Byte level access to the Phrase */
     Struct
     {
     UINT8 PhraseSize; /* Phrase size (wihtout including the size on the count) */
     UINT8 PhraseAddress[3]; /* Address, on S12Z devices its 24-bit addressing */
     UINT8 PhraseData[33]; /* Maximum 32 data bytes. The last byte is the CRC */
     }F;
}BootPhraseStruct;
```

引导加载应用程序创建一个 BootPhraseStruct 结构的实例，称为“BP”。一旦完成了 S2 行的接收并存放于“BP”结构，通信驱动程序必须将“u8BootPhraseRcvd”标志置 1。请注意，“BP”和“u8BootPhraseRcvd”为外部变量，通信驱动程序必须对其进行修改。引导加载程序将检查“u8BootPhraseRcvd”标志是否置 1，如果置 1，将清零该标志并处理“BP”上的接收数据，以便写入。数据写入完成后，引导加载程序将调用 Comms_Tx_Ack 函数，指示 S-Record 发送器发出新的 S2 行。请注意，“BP”结构无需完全填满也能处理。S2 记录可以是不同大小。引导加载程序将只使用 S2 记录本身指示的数据大小（字节大小)。通信驱动程序必须通过“大小”字段检查当前 S2 行是否完成发送。发送器不会发出信号来指示行发送完成。

4.3 存储器擦除

当接收到第一个 S2 行时，会验证校验和，如果匹配则擦除 P-Flash 存储器。调用擦除 P-Flash 函数将擦除 P-Flash 上的所有扇区（引导加载程序所在的扇区除外）。擦除过程包括最后一个扇区，其中存放存储器的保护字节、中断和启动向量。引导加载程序必须放置在 P-Flash 阵列的后几个扇区（在最后一个扇区之前）。下表说明了扇区的用途。

表 5. 引导加载程序在 P-Flash 扇区的布局

起始地址	结束地址	用途	大小[字节]
...	...	...	...
0xFFF800	0xFFF9FF	引导加载程序。可从 Flash 执行的函数	1024
0xFFFA00	0xFFFBFF		
0xFFFC00	0xFFFDFF	引导加载程序。可从 RAM 执行的函数	512
0xFFFE00	0xFFFFFF	中断和启动向量。P-Flash 存储器保护位	512

引导加载程序分为两大部分：必须从 RAM 执行的函数和可从 Flash 执行的函数。必须从 RAM 执行的函数是 P-Flash 擦除和 P-Flash 写入。其他代码，包括通信驱动程序，均可从 Flash 执行。当包含小型 SCI 通信驱动程序时，引导加载程序的大小为 1.5 KB。该大小会随所采用的通信驱动程序大小而变化。

在写入任何信息之前，位于引导加载程序前后的所有 P-Flash 扇区均被擦除。常规扇区（位于引导加载程序之前的扇区）无需特别注意，最后一个扇区需要进行部分重编程：

- P-Flash 保护字节：当擦除最后一个扇区时，这些字节默认为 0xFF。将 FSEC 字节保留为 0xFF 将在下次 MCU 复位时锁住器件。其余保护字节可保留为 0xFF，这意味着未使能写保护。擦除 P-Flash 函数在擦除最后一个扇区之前先保存这些保护字节，并在之后对其进行重写。这保证 MCU 不是被无意加锁。
- 启动向量：当擦除最后一个扇区时，启动向量默认为 0xFFFFFF。将启动向量保留为该地址会阻止 MCU 运行任何程序，包括启动引导加载程序。需要通过外部重编程来恢复 MCU。尽快将启动向量重编程为引导加载程序的启动地址是关键。擦除 P-Flash 例程在擦除最后一个扇区之前先保存引导加载程序的启动地址，并在之后立即重编程启动向量。

注

从擦除最后一个 P-Flash 扇区到重编程启动向量为引导加载程序启动地址之间存在一个 20.5 ms 的窗口。用户必须确保在此期间不能断电，否则，下次复位时 MCU 将不能运行。MCU 需通过外部重编程才能重新运行。

在 P-Flash 扇区擦除且启动向量重编程后，MCU 将返回到在 Flash 中执行的常规流程，并准备写入第一个 S2 行。

4.4 Flash 编程

在接收 S2 记录且验证其校验和有效后，引导加载程序将准备第一次写操作。如章节 3“S-Record 格式”中所述，每个 S2 行由 3 个地址字节组成，最多 32 个数据字节。由于 FMTRZ 一次最多（且最少）写入 8 个字节，一条 S2 记录可分为 4 条写指令。每条 P-Flash 写指令必须与 8 字节边界对齐才能成功完成。这意味着 FTMRZ 将只写入最后 3 个位为 0 的地址。每条写指令将从该地址开始写入 8 个连续字节。

例如：用户希望将 0xAC 写入地址 0xFFFE12

这里，直接写入地址 0xFFFE12 会发生错误，因为该地址不与 8 字节边界对齐（最后 3 位为 0b010）。用户必须先找到与存储器地址相对应的 8 字节对齐地址，这里为 0xFFFE10。通过 8 字节边界对齐地址，用户可以将任意 8 字节数据连续写入该地址。在这里，用户需要写入：

该存储器写操作修改了以目标地址为首的所有 8 个连续字节的值。这里，所有字节均写为 0xFF。写入后，只有擦除整个扇区才能修改这些字节。

在引导加载应用程序中，该过程自动执行，并且 BootPhraseStruct 中的数据按字节传输到一个称为“ProgramStruct”的更小的结构中。ProgramStruct 包含一个 8 字节对齐的 32 位地址，以及必须写入该地址的 8 数据字节：

```
typedef struct
{
         UINT8 Data[8];
         union
         {
             UINT8 Byte[4]; /* Byte level access to the Address */
             UINT32 DWord; /* DWord level access to the Address */
         } Add;
}ProgramStruct;
```

引导加载应用程序使用 ProgramStruct 中称为 PS 的实例。P-Flash 编程例程利用 PS 结构将数据写入 Flash。P-Flash 编程例程必须从 RAM 执行，与 P-Flash 擦除例程一样。一旦编程完成，将返回执行 Flash 代码，并将下一个数据填入 PS。在 PS 结构为满（第 8 个字节值被加载）或 BootPhraseStruct 数据完成时会调用编程例程。所有不具有确切值的字节均编程为 0xFF。

引导加载应用程序尝试写入最后一个地址 (启动向量) 时，将认为编程已完成。将忽略最后的写操作，因为在 P-Flash 擦除后，会立即使用引导加载程序启动地址对启动向量进行重编程。除非擦除整个扇区，否则无法对该地址进行重编程。

4.5 跳转到主应用程序

引导加载应用程序在两种情况下会跳转到主用户应用程序：超时或尝试写入启动向量。复位时，MCU 会访问启动向量，寻找入口函数的地址。由于该地址总是通过引导加载程序进行重编程，因而跳转到主应用程序通过创建一个指向可编程固定地址的指针来完成。在跳转到主应用程序之前，需要执行一些管理活动。

为了确保主应用程序以已知和预期的状态接收 MCU，引导加载程序必须将所有修改的寄存器返回其初始值。引导加载程序代码将 CPMU、RTI、FTMRZ 和 GPIO 中的寄存器返回为其初始值。通信驱动程序必须如数据接收 中描述的那样，在“Comms_Reset_Reg”例程中实现该寄存器复位功能。

在通信驱动程序复位寄存器例程返回后，MCU 会跳转到预编程的指针并开始执行代码。下一章节将说明如何配置启动地址和许多其他自定义内容。

5 实施 MagniV 引导加载程序

MagniV 引导加载程序设计为可升级且易于在具有 S12Z 内核的器件之间移植。当将引导加载应用程序移植到一个新的 S12Z 器件时，用户必须注意三个方面：

- 设置链接器文件的地址和扇区
- 在“main.h”文件上查看和设置引导加载程序的配置宏
- 实施或移植通信驱动程序

5.1 文件结构

引导加载程序（本应用说明随附）的源文件分别位于两个文件夹：Bootloader 和 Comms。“Bootloader”文件夹包含实际数据处理、存储器擦除和编程所需的全部文件。“Comms”文件夹包含一些通信驱动程序示例。本文件夹中包含通用“comms.h”文件和用于不同模块和协议的通信驱动程序的专用文件夹。不能修改“comms.h”文件夹，因为其中定义了引导加载程序和通信驱动程序之间的接口函数。“comms.c”文件是数据接收和发送驱动程序的实际设置。用户必须根据具体应用的通信协议、通道和其他特殊配置来设置该文件。图 5 是 CodeWarrior 项目中引导加载应用程序的文件结构示例。

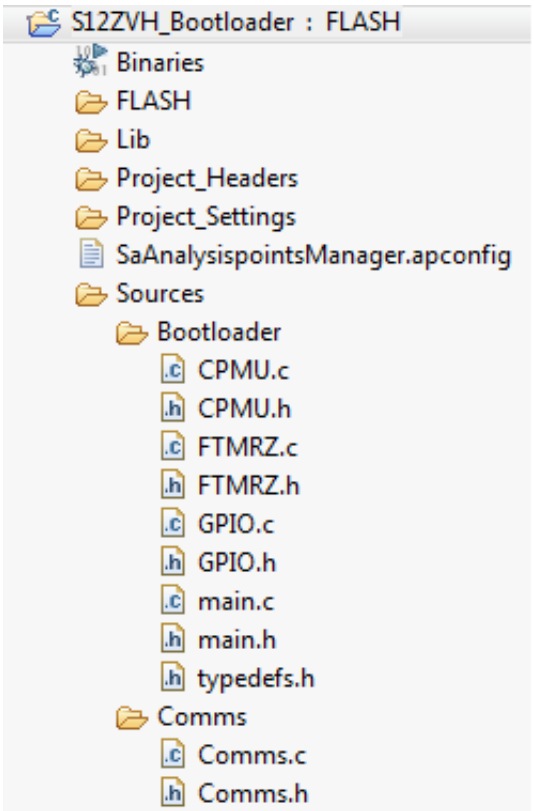

图 5. S12Z MagniV 引导加载程序结构示例

Comms 文件夹保留用于通信驱动程序，可以不止两个文件。 唯一限制是“Comms.h”中定义的函数必须在通信驱动程序中实施。

5.2 链接器文件配置

必须修改引导加载应用程序中的链接器文件。新的链接器文件必须确保将引导加载程序代码分为两组，一组用于从 Flash 执行的函数，另一组用于从 RAM 执行的函数。这些分别称为 ROM 和 SHADOW_ROM_S。必须修改这些扇区的地址，使其位于 P-Flash 阵列的最后（位于中断向量之前），请参见表 5。SHADOW_ROM_S 必须为一个扇区大小 (512 字节)，而 ROM 的大小由通信驱动程序大小决定（默认为 1k)。

RAM 存储器也需要分割, 因为 SHADOW_ROM_S 上的函数也需要复制到 RAM 上。需要创建 SHADOW_RAM_S 扇区，且必须与其相对应的 ROM 大小一致（512 字节)。链接器文件（CodeWarrior 中的.prm 文件）应采用以下布局:

```
NAMES END
SEGMENTS
  SHADOW_RAM_S = READ_WRITE 0x001000 TO 0x0011FF; // 512. Functions executed on RAM
  RAM = READ_WRITE 0x001200 TO 0x0013FF; // Rest for general RAM purposes

  EEPROM = READ_ONLY 0x100000 TO 0x10007F;

  ROM = READ_ONLY 0xFFF800 TO 0xFFFBFF; // 1K. Functions executed on Flash
  SHADOW_ROM_S = READ_ONLY 0xFFFC00 TO 0xFFFDFF; // 512. Functions executed on RAM
END

PLACEMENT
  _PRESTART, /* Used in HIWARE format: jump to _Startup at the code start*/
  STARTUP, /* startup data structures */
  ROM_VAR, /* constant variables */
  STRINGS, /* string literals */
  VIRTUAL_TABLE_SEGMENT, /* C++ virtual table segment */
```

```
  NON_BANKED, /* runtime routines which must not be banked */
  DEFAULT_ROM,
  COPY INTO ROM;
  SHADOW_ROM INTO SHADOW_ROM_S;

  SSTACK, /* allocate stack first to avoid overwriting variables on
  DEFAULT_RAM INTO RAM;
  SHADOW_RAM INTO SHADOW_RAM_S;
END

ENTRIES
END

STACKSIZE 0x100

VECTOR 0 _Startup /* reset vector: this is the default entry point for a C/C++ application.
```

请注意，PLACEMENT 部分新增两行，一行将 SHADOW_ROM 映射到 SHADOW_ROM_S，另一行将 SHADOW_RAM 映射到 SHADOW_RAM_S。这确保相应的变量和函数存放在对应的存储器扇区内。

5.3 引导加载程序配置宏

一旦配置了链接器文件，用户可以继续在“main.h”文件中更新引导加载应用程序的宏。根据每款器件和应用程序，共有 11 个宏可用于配置引导加载应用程序。这些宏可以分为 8 组：

- 振荡器频率：
  - OSCILLATOR_FREQUENCY：该宏指示目标电路板上的振荡器频率（MHz）。可用值有 4、8、12 和 16。该值用于将 PLL 配置为 32 MHz BusClk。
- 超时：
  - TIMEOUT：这是引导加载程序在 POR 后等待有效 S-Record 的时间。该值以 100 毫秒为单位。TIMEOUT 为 1 则在跳转到主应用程序前会等待 100 ms。
- 擦除 Flash 宏：
  - FLASH_START_ADD：这是 P-Flash 阵列的首字节地址（十六进制）。
  - FLASH_SECTORS_TO_ERASE：这是要擦除的扇区数。考虑一个 Flash 扇区为 512 字节。这里不计算包含引导加载程序和中断向量的扇区。

    FLASH_SECTORS_TO_ERASE = Total_Sec – SR_Sec – R_Sec – IntVec_Sec 其中：**Total_Sec**：P-Flash 上的总扇区数。一个扇区为 512 字节。**SR_Sec**：SHADOW_ROM_S 分段上的扇区数。必须为 1。 **R_Sec**：ROM 分段上的扇区数。默认为 2，不过可能会变。**IntVec_Sec**：中断向量的扇区数。必须为 1。
- 影子分段宏
  - SHADOW_ROM_ADD：这是影子 ROM 扇区的首字节地址（十六进制）。
  - SHADOW_RAM_ADD：这是影子 RAM 扇区的首字节地址（十六进制）。
- 用于视觉指示的 GPIO 宏
  - ACTIVITY_LED_ENABLE：使能/禁止 LED 在每次接收到 S2 记录时翻转。用于指示活动。可能值为 1 或 0。如果为 0（禁止），将无需理会接下来的 GPIO 宏。
  - ACTIVITY_LED_DDR：指向用作活动指示 LED 的数据方向寄存器。例如 DDRP_DDRP0。
  - ACTIVITY_LED：指向用作活动指示 LED 的数据寄存器。例如 PTP_PTP0。
  - LED_ON：点亮 LED 的数值。1 用于 LED 点亮配置，0 用于 LED 关闭配置。
- 校验和检查
  - CHECK_PHRASE_CHECKSUM：启用/禁止校验和计算并比较例程。在通信协议中禁止校验和以确保数据传送。当禁止 CHECK_PHRASE_CHECKSUM 时，检查例程总是返回 OK。
- 应用程序启动向量
  - APPLICATION_START_ADD：主应用程序启动向量地址。当发生超时或新程序加载到 P-Flash 时，引导加载程序将跳转到该地址。一旦引导加载程序加载到目标器件后，将不能修改该地址 (除非在目标器件上重新编译和编写引导加载程序)。加载在该目标器件上的所有应用程序必须从该地址开始才能运行。通常建议启动地址为 P-Flash 的首地址。

6 开发用于 MagniV 引导加载程序的代码

一旦引导加载应用程序加载到目标器件上，它将可以接收 S-Record 文件并多次重编程 P-Flash。一旦新程序写入 P-Flash 且引导加载程序准备跳转到主应用程序时，将释放所有 MCU 资源。主应用程序可以访问所有 RAM 存储器和 MCU 外设。

设计在具有 MagniV 引导加载程序的 MCU 上运行的应用程序应注意以下两个方面：

- 不要覆盖引导加载程序：用户必须确保引导加载程序所在的扇区不被主应用程序使用。这可通过在主应用程序的链接器文件中修改 ROM 分段来避免。将 ROM 分段的结束地址设在引导加载程序的首个扇区之前。
- 将启动函数地址与预编程的启动地址相匹配：主应用程序的入口地址必须与已装载到目标器件上的引导加载程序中的一个预编程地址相匹配。通常建议引导加载程序使用 P-Flash 的首个地址作为入口点，虽然 P-Flash 上的任意地址均能预编程为主应用程序的启动地址。

未能满足这些要求不会损坏或擦除目标器件上的引导加载程序。只是主应用程序将不能正常运行，但其总能通过加载新的 S-Record 文件来重编程。如果用户希望从主应用程序启动引导加载程序的执行，建议通过软件强制复位。这通常通过配置看门狗并提供一个不正确的复位值来完成。看门狗模块将立即触发一个复位，随后引导加载应用程序将开始执行。

一旦应用程序就绪，需要对其进行编译并将其链接生成 S-Record 文件。大多数编译器在编译后可以直接生成 S-Record 文件。用户必须确保在编译器配置上启用 S-Record 生成。

7 PC 应用程序编程

本应用说明随附的内容当中包含可用于通过 RS-232（SCI）传输 S-Record 文件的 PC 应用程序。该应用程序命名为 S12Z MagniV Serial Bootloader Interface。它可以选择用于通信的 COM 端口，以及指定 S-Record 文件的路径。任何文本文件均可用作输入文件，因为不会检查扩展名。请注意，S12Z MagniV 引导加载程序仅需要 S2 行，因此应用程序将只发送以“S2”开头的文本行，其他行将被忽略。

PC 应用程序需要来自目标的应答，以指示上次的发送行是否正确处理或是否检测到校验和错误。如果发生校验和错误，将重新发送该行。当一行被正确接收后，会发送下一行。

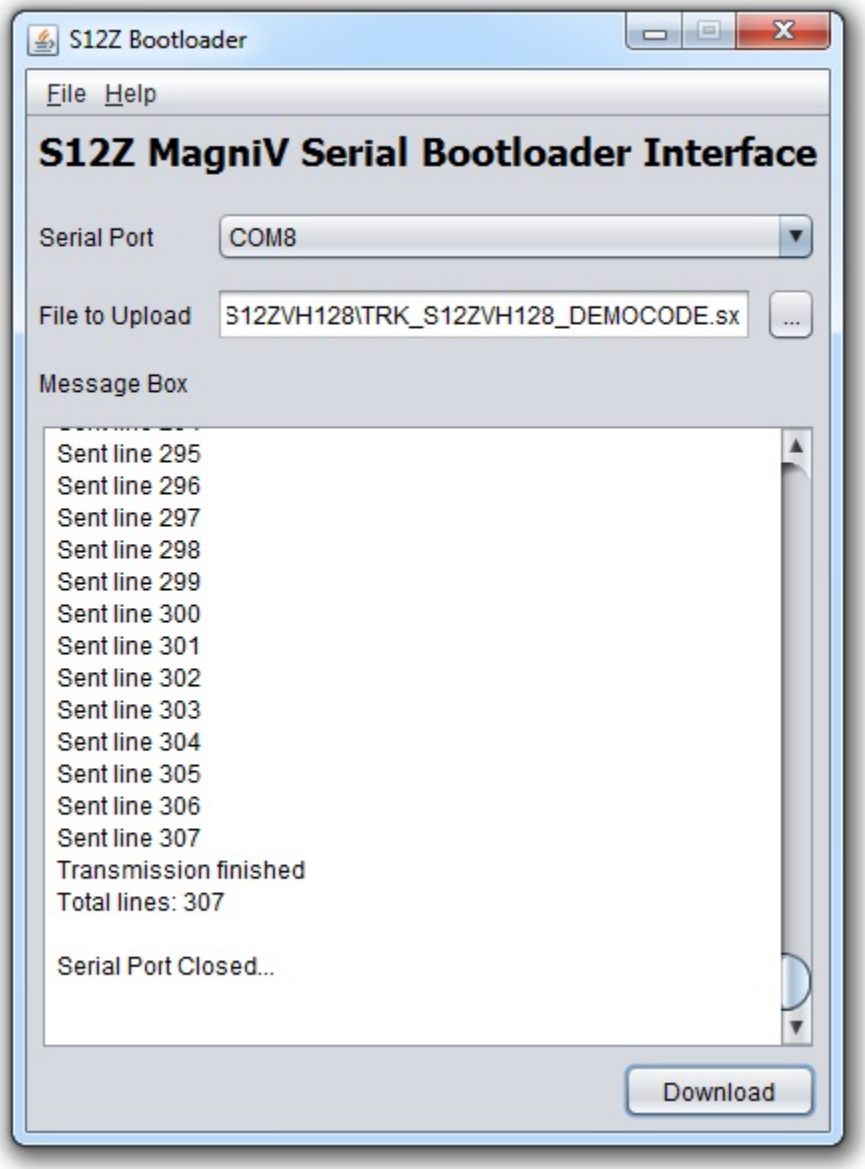

图 6. S12Z MagniV Serial Bootloader Interface 的截屏

S12Z MagniV Serial Bootloader Interface 配置为在 19,200 波特率下工作，具有一个停止位，无奇偶校验。这些配置不是用户可选的。该应用程序在 Java 上开发，需要 Java 运行时环境（JRE）才能运行。目前该应用程序仅在 Windows 操作系统上运行，因为其使用 Microsoft 开发的 win32com.dll。

8 结论

S12Z MagniV 引导加载应用程序是可扩展、交叉平台项目，帮助用户简单快速实现用于其应用程序的引导加载程序。引导加载程序和通信驱动程序的明确分开为在多个不同应用程序和器件上实现移植提供了灵活性，无需大量修改代码。引导加载程序在器件上运行时仅需低于 1 K 的 RAM 空间及平均 1.5 K 的 ROM 空间。不使用时不会消耗 MCU 资源，将完全访问交于主用户应用程序。该软件应用程序和 PC 接口程序由飞思卡尔提供。仅作为演示代码用于开发目的。分享的代码不着眼于成为生产级别的应用程序，飞思卡尔不保证在这种情况下能工作正常。

9 参考

请参考最新的 MagniV 器件、新闻、资源、参考设计和其他信息：http://www.freescale.com/MagniV

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN4723
Revision 0, 07/2013

freescale™