应用笔记

Application Note

文档编号：**AN1137**

G32R501 BootLoader 用户手册

版本：**V1.0**

1. 引言

此参考手册适用于存储在 R501 系列处理器的片上引导 ROM 中的代码和数据，其中包括基于 FLASH、基于 ROM 和基于 RAM 的器件。

引导 ROM 在出厂时已设定好引导加载软件。引导模式信号（通用的 I/O）用于指示 bootloader 软件在上电时要使用哪种模式。

本手册描述了 bootloader 的用途和特性，它还描述了器件的片上引导 ROM 的其它内容，并标识了所有信息在该存储器内的位置。

目录

2. 引导 ROM 概述

2.1. 引导 ROM 存储器映射

引导 ROM 是位于地址 0x10000000-0x10020000 的 128KB 只读存储器块。

- 引导加载程序功能
- Flash API 库
- Fix32math 表
- 版本号

表格 1 ROM 地址分配

模块	起始地址	结束地址	大小
Boot	0x10000000	0x10003F70	16240 Byte
FlashAPI	0x10004000	0x10008000	16384 Byte
AES_Table	0x1001AE00	0x1001D200	9216 Byte
Fix32mathTables	0x1001D600	0x1001DE04	2052 Byte
CFFT_f32_twiddleFactors	0x1001E300	0x1001EF00	3072 Byte
RFFT_f32_twiddleFactors	0x1001EF00	0x1001FEF0	4080 Byte
CRCTable	0x1001FFA0	0x1001FFAF	16 Byte
Version	0x10003FFC	0x10004000	4 Byte

3. Boot 特性

3.1. Boot 模式

本节介绍 G32R501 支持的 Boot 模式。

Boot ROM 使用引导控制的 GPIO 引脚来确定引导模式配置。设备可以配置为引导到 RAM、引导到 Flash、执行引导加载程序或保持在等待模式中。

表格 2 显示了默认的引导模式选项，用户可以自定义支持的引导模式以及引导模式选择引脚。

表格 2 G32R501 默认 Boot 模式

Boot 模式	**GPIO24** **（默认模式下 Pin1）**	**GPIO32** **（默认模式下 Pin0）**
Parallel IO	0	0
UART / Wait boot	0	1
CAN	1	0
Flash	1	1

表格 3 是 G32R501 支持的所有 Boot 模式，用户可以根据项目需要自定义所需引导模式，以及引导模式选择引脚。

表格 3 G32R501 所有可用的 Boot 模式

Boot 模式序号	**Boot 模式**
0	Parallel IO
1	UART / Wait boot
2	CAN
3	Flash
4	Wait
5	RAM
6	SPI Master
7	I2C Master
10	Secure Boot

3.2. Boot 模式的 GPIO 引脚分配

用户可以根据封装及其他配置信息来选择不同的 GPIO 引脚。

表格 4 UART Boot

配置	Boot 模式配置值	UART Tx 引脚	UART Rx 引脚
0	0x01	GPIO29	GPIO28
1	0x21	GPIO16	GPIO17
2	0x41	GPIO8	GPIO9
3	0x61	GPIO48	GPIO49
4	0x81	GPIO24	GPIO25

表格 5 CAN Boot

配置	Boot 模式配置值	UART Tx 引脚	UART Rx 引脚
0	0x02/82	GPIO32	GPIO33
1	0x22/A2	GPIO4	GPIO5
2	0x42/C2	GPIO31	GPIO30
3	0x62/E2	GPIO37	GPIO35

表格 6 Flash Boot

配置	Boot 模式配置值	Flash 启动地址	Flash 块
0	0x03	0x08000000	Bank 0, Sector 0（Busmatrix IF）
1	0x23	0x00100000	Bank 0, Sector 0（ITCM IF）
2	0x43	0x08020000	Bank 1, Sector 0 （Busmatrix IF,DualBank Mode）
3	0x63	0x00120000	Bank 1, Sector 0 （ITCM IF,DualBank Mode）

表格 7 Wait Boot

配置	Boot 模式配置值	看门狗状态
0	0x04	Enable
1	0x24	Disable

表格 8 RAM Boot

配置	Boot 模式配置值	RAM 启动地址	RAM 属性
0	0x05	0x00000000	ITCM
1	0x25	0x20100000	SRAM0
2	0x45	0x20200000	SRAM1
3	0x65	0x20300000	SRAM2

表格 9 SPI Boot

配置	Boot 模式配置值	MOSI	MISO	SCK	CS
1	0x26	GPIO8	GPIO10	GPIO9	GPIO10
2	0x46	GPIO54	GPIO55	GPIO56	GPIO57
3	0x66	GPIO16	GPIO17	GPIO56	GPIO57
4	0x86	GPIO8	GPIO17	GPIO9	GPIO11

表格 10 I2C Boot

配置	Boot 模式配置值	SDA	SCL
0	0x07	GPIO32	GPIO33
1	0x47	GPIO26	GPIO27
2	0x67	GPIO42	GPIO43

表格 11 Secure Boot

配置	Boot 模式配置值	Flash 启动地址	Flash 块
0	0x0A	0x08000000	Bank 0, Sector 0（Busmatrix IF）
1	0x2A	0x00100000	Bank 0, Sector 0（ITCM IF）
2	0x4A	0x08020000	Bank 1, Sector 0（Busmatrix IF,DualBank Mode）
3	0x6A	0x00120000	Bank 1, Sector 0（ITCM IF,DualBank Mode）

3.3. BootMode 状态

BootMode 为 Selectboot 运行结果，标志了当前选择的启动项。

基地址：0x2030_7F00

结构：

表格 12 当前选择的 Boot 模式

位	名称	描述
[31:8]	RESERVE	保留
[7:5]	Boot Option	用于指示当前启动方式的配置
[4:3]	RESERVE	保留
[2:0]	Boot Mode	用于指示当前启动方式： 0 : Parallel IO 1 : UART / Wait boot 2 : CAN 3 : Flash 4 : Wait 5 : RAM 6 : SPI Master 7 : I2C Master 10 : Secure Boot

3.4. Boot 执行进度信息

用户可以通过该寄存器置位信息来确认 Boot 执行过程中的进度信息。

基地址：0x2030_7F08

结构：

表格 13 Boot 执行进度信息

位	状态	描述
31	BOOTROM_IN_HARDFULT_HANDLER	Boot 进入 Hard Fault 处理函数
[30:28]	/	RESERVE
27	BOOTROM_GOT_A_HARD_FAULT	Boot 产生了 Hard Fault
26	BOOTROM_GOT_A_USAGE_FAULT	Boot 产生了 Usage Fault
25	BOOTROM_GOT_A_BUS_FAULT	Boot 产生了 Bus Fault
24	BOOTROM_GOT_A_MEM_FAULT	Boot 产生了 Memory Management Fault
23	/	RESERVE
22	BOOTROM_GOT_A_MCLK_NMI	Boot 产生了时钟丢失 NMI
21	/	RESERVE
20	BOOTROM_GOT_A_FLASH_UNCERR_NMI	Boot 产生了 Flash ECC 不可纠正错误 NMI
[19:16]	/	RESERVE
15	BOOTROM_BOOT_COMPLETE	启动完毕
14	/	RESERVE
13	BOOTROM_HANDLED_POR	Boot 正在清除 POR 标志位
12	BOOTROM_HANDLED_XRSN	Boot 正在清除 XRSN 标志位
11	BOOTROM_RESC_HANDLED	Boot 正在处理复位标志
10	BOOTROM_POR_MEM_TEST_DONE	Boot 已完成 MBIST
9	/	RESERVE
8	BOOTROM_IN_SECURE_BOOT	Boot 进入 Secure BOOT
7	BOOTROM_IN_CAN_BOOT	Boot 进入 CAN BOOT
6	BOOTROM_IN_I2C_BOOT	Boot 进入 I2C BOOT
5	BOOTROM_IN_SPI_BOOT	Boot 进入 SPI BOOT
4	BOOTROM_IN_UART_BOOT	Boot 进入 UART BOOT
3	BOOTROM_IN_RAM_BOOT	Boot 进入 RAM BOOT
2	BOOTROM_IN_PARALLEL_BOOT	Boot 进入 PARALLEL IO BOOT
1	BOOTROM_IN_FLASH_BOOT	Boot 进入 FLASH BOOT
0	BOOTROM_START_BOOT	开始启动流程

3.5. Boot 执行异常信息

表格 14 Boot 执行异常信息

异常地址	异常名称	描述
0x20307F0C	hard_fault_status	记录进入 Hard Fault 时的 SCB_HFSR 状态
0x20307F10	cbrom_fault_status	记录进入 NMI 时的 NMI_FLG
0x20307F14	bus_fault_adress	记录进入 Hard Fault 时的 SCB_BFAR 状态
0x20307F18	bus_fault_status	记录进入 Hard Fault 时的 SCB_BFSR 状态
0x20307F1C	mem_manage_fault_address	记录进入 Hard Fault 时的 SCB_MMFAR 状态
0x20307F20	mem_manage_fault_status	记录进入 Hard Fault 时的 SCB_MMFSR 状态
0x20307F24	usage_fault_status	记录进入 Hard Fault 时的 SCB_UFSR 状态

3.6. Boot Wait Point

在启动 ROM 执行期间，CPU 可能会在代码中进入等待循环状态。这种状态可能由于多种原因而发生。**错误!未找到引用源。**列出了 CPU 程序计数器（PC）寄存器值落入的地址范围，用户可以通过读取当前 PC 地址来判断 Boot 是否进入异常。

表格 15 Boot 等待点地址信息

等待点类型	等待点地址
In Wait Boot	0x1000_17DC—0x1000_17F4
In UART Boot	0x1001_1DC4—0x1001_1E14
In Hard Fault Handler	0x1000_0212—0x1000_0312
In NMI Handler	0x1000_0316—0x1000_044E
Failed secure Flash CMAC verification loop	0x1001_0F80—0x10010FB8

4. BootLoader 特性

本部分详细描述了引导模式选择进程以及 bootloader 操作的具体细节。

4.1. BootLoader 器件配置

引导 ROM 代码在执行过程中访问多个内存地址和寄存器。有两组地址，一组用于仿真，另一组用于独立引导流程。仿真位置模拟 OTP 配置，可以根据需要写入多次。用户可配置的 DCSM OTP 位置用于独立引导流程，程序将设备 OTP，因此只能写入一次。表格 16 为仿真引导和独立引导的寄存器及配置地址。

表格 16 Boot 寄存器地址

Boot 模式	DCSM 名称	Boot ROM 名称	地址
Emulation	Z1-GPREG1	EMU-BOOTPIN-CONFIG	0x50020000
	Z1-GPREG3	EMU-BOOTDEF-LOW	0x50020004
	Z1-BOOTCTRL	EMU-BOOTDEF-HIGH	0x50020008
Standalone	Z1-GPREG1	Z1-OTP-BOOTPIN-CONFIG	0x0810A018
	Z1-GPREG2	Z1-OTP-BOOT-GPREG2	0x0810A01C
	Z1-GPREG3	Z1-OTP-BOOTDEF-LOW	0x0810A038
	Z1-BOOTCTRL	Z1-OTP-BOOTDEF-HIGH	0x0810A03C

执行设备初始化后，引导加载程序将检查 TRST 引脚的状态，以确定是否连接了仿真器。

- 仿真启动（仿真器已连接且/TRST=1）

在仿真启动中，启动 ROM 将检查三个寄存器位置，称为 EMU_BOOTPIN_CONFIG，EMU_BOOTDEF_LOW 和 EMU_BOOTDEF_HIGH 作为启动模式。如果任意位置的内容无效，则使用“等待”启动模式。在执行仿真引导时，可以通过调试器修改 EMU_BOOTDEF 的值来访问所有引导模式选项。

- 独立启动（仿真器未连接且/TRST=0）

如果设备处于独立启动模式，默认情况下，使用两个 GPIO 引脚的状态来确定将执行哪种启动模式。选项包括:Parallel IO、UART/WAIT、CAN 和 FLASH。每个模式都在中详细描述。但可以通过在 OTP 中编程两个值以选择另一个引导加载程序来自定义。这里提到的这些启动模式在第 3.2 节中有详细描述。

在选择过程之后，如果所需的引导加载完成，处理器将在由所选引导模式确定的入口点继续执行。如果调用了引导加载程序，则外围设备加载的输入流将确定此条目地址。第 4.4 节描述了该数据流。相反，如果您选择直接启动到 Flash 或者 SARAM，则条目地址是为每个内存块预定义的。以下各节详细讨论了可用不同的引导模式以及用于将数据代码加载到设备中的过程。

4.2. BootLoader 模式

表格 17 解释了如何通过编程用户可配置的 DCSM OTP 中的 BOOTPIN_CONFIG 位置来定制引导模式选择引脚。DCSM OTP 中的位置为 Z1-OTP-BOOTPIN-CONFIG。在调试时，EMU-BOOTPIN-CONFIG 是 Z1-OTP-BOOTPIN-CONFIG 的仿真等效项，可以进行编程以实验不同的引导模式，而无需写入 OTP。设备可以根据需要编程使用 0、1、2 或 3 个引导模式选择引脚。

表格 17 BOOTPIN_CONFIG 寄存器

位	名称	描述
31-24	Key	将 0x5A 写入这 8 位，以告知 boot ROM 代码该寄存器中的位是有效的。
23-16	Boot Mode Select Pin 2（BMSP2）	参考 BMSP0 描述。
15-8	Boot Mode Select Pin 1（BMSP1）	参考 BMSP0 描述。
7-0	Boot Mode Select Pin 0（BMSP0）	将启动的 GPIO 引脚设置为（最多 255 个）。0x0 = GPIO0；0x01 = GPIO1，以此类推，0xFF 无效，并选择默认配置的 BMSP0，如果所有其他 BMSP 也设置为 0xFF。任何其他 BMSP 未设置为 0xFF，则将某个 BMSP 设置为 0xFF 将禁用该特定 BMSP。

本节解释了如何为设备配置引导定义表（BOOTDEF）及相关引导选项。该 64 位位置位于用户可配置的 DCSM OTP 中，具体为 Z1-OTP-BOOTDEF-LOW 和 Z1-OTP-BOOTDEF-HIGH 位置。在调试时，EMU-BOOTDEF-LOW 和 EMU-BOOTDEF-HIGH 是 Z1-OTP-BOOTDEF-LOW 和 Z1-OTP-BOOTDEF-HIGH 的仿真等效项，可以进行编程以实验不同的引导模式选项，而无需写入 OTP。引导定义表的自定义范围取决于使用的引导模式选择引脚数量。

表格 18 BOOTDEF 寄存器

BOOTDEFx	位	名称	描述
BOOT_DEF0	7-0	BOOT_DEF0 Mode/Options	设置启动模式和启动模式选项。这可以包括更改特定启动外设的 GPIO 或指定不同的 Flash 入口点。任何不支持的启动模式都会导致设备重置。有关有效的 BOOTDEF 值，请参阅第 3.2 节。
BOOT_DEF1	15-8	BOOT_DEF1 Mode/Options	参考 BOOT_DEF0 描述。
BOOT_DEF2	23-16	BOOT_DEF2 Mode/Options	
BOOT_DEF3	31-24	BOOT_DEF3 Mode/Options	
BOOT_DEF4	39-32	BOOT_DEF4 Mode/Options	
BOOT_DEF5	47-40	BOOT_DEF5 Mode/Options	
BOOT_DEF6	55-48	BOOT_DEF6 Mode/Options	
BOOT_DEF7	63-56	BOOT_DEF7 Mode/Options	

下面演示了配置启动模式选择引脚的一些用例。

- 无启动模式选择引脚

该用例演示了一个不使用任何启动模式选择引脚的应用场景，设备始终从 Flash 启动。

按如下设置在 OTP 中编程 BOOTPIN_CONFIG 位置：

（1） 将 BOOTPIN_CONFIG.BMSP0 设置为 0xFF

（2） 将 BOOTPIN_CONFIG.BMSP1 设置为 0xFF

（3） 将 BOOTPIN_CONFIG.BMSP2 设置为 0xFF

（4） 将 BOOTPIN_CONFIG.KEY 设置为 0x5A，以便启动 ROM 将这些寄存器位视为有效。

（5） 为设备编程 BOOTDEF 位置选项。这基本上设置了一个设备特定的启动模式表。

（6） 将 BOOTDEF.BOOTDEF0 设置为 0x03，以从 Flash 启动，启动模式值为 0。

表格 19 无启动模式选择引脚

Boot 模式序号	Boot 模式
0	Flash Boot（0x03）

- 一个启动模式选择引脚

该用例演示了一个使用单个启动模式选择引脚的应用场景，以选择从 Flash 启动或使用 CAN 启动。

按如下设置在 OTP 中编程 BOOTPIN_CONFIG 位置：

（1） 将 BOOTPIN_CONFIG.BMSP0 设置为用户指定的 GPIO，例如 0x0 表示 GPIO0

（2） 将 BOOTPIN_CONFIG.BMSP1 设置为 0xFF

（3） 将 BOOTPIN_CONFIG.BMSP2 设置为 0xFF

（4） 将 BOOTPIN_CONFIG.KEY 设置为 0x5A，以便启动 ROM 将这些寄存器位视为有效。

（5） 为设备编程 BOOTDEF 位置选项。这基本上设置了一个设备特定的启动模式表。

（6） 将 BOOTDEF.BOOTDEF0 设置为 0x02，以使用启动模式值 0 从 CAN 启动。

（7） 将 BOOTDEF.BOOTDEF1 设置为 0x03，以使用启动模式值 1 从 Flash 启动。

表格 20 一个启动模式选择引脚

Boot 模式序号	Boot 模式
0	CAN Boot（0x02）
1	Flash Boot（0x03）

参阅第 3.2 节可获取其他 Boot Mode 配置。

4.3. BootLoader 函数操作

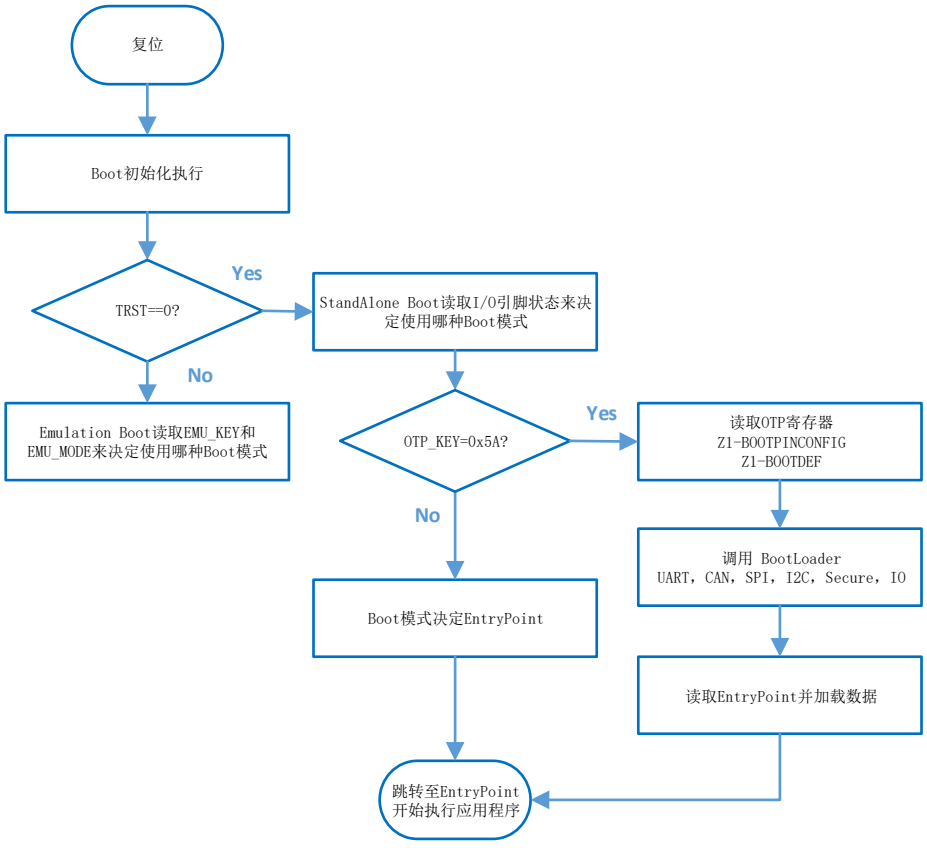
Bootloader 是位于片上引导 ROM 中的在复位后执行的程序。

Bootloader 用于在上电后将代码从外部源传输到内存储器中；从而允许将代码驻留在外部的慢速非易失存储器中，然后传输至高速存储器中执行。

Bootloader 提供了多种不同的下载代码方式以适应不同的系统要求。Bootloader 使用各种 GPIO 信号确定将使用哪种引导模式。本文档中的其余部分描述了引导模式选择进程以及每个 bootloader 的具体细节。

图 1 显示了基本 bootloader 流程。

图 1 Bootloader 执行流程



4.4. BootLoader 数据流结构

表格 21 和相关示例显示了引导加载程序的传入数据流的结构。数据流结构中的所有值均采用十六进制。数据流中的第一个 16bit 键值，即 0x08AA。如果引导加载程序收到无效的键值，则加载将中止。接下来的十六个字节用于初始化寄存器值，或者通过向引导程序传递值来增强引导程序。如果引导加载程序不使用这些值，则将它们保留以备将来使用，引导加载程序仅读取该值，然后将其丢弃。只有 SPI，I2C 和 CAN 引导加载程序使用这些词来初始化寄存器。第十九个、第二十个、第二十一个和第二十二个字节包括 22 位入口点地址。该地址用于在启动加载完成后初始化 PC。该地址很可能是引导加载程序下载的程序的入口点。数据流中的第二十三个、第二十四个字节是要传输的第一个数据块的大小。例如，要传输 20 个 8 位数据值的块，块大小将为 0x000A 以指示 20 个 8 位字节。接下来的四个字节加载程序指示数据块的目标地址。对于要传输的每个数据块，此块大小/目标地址模式重复。传输完所有块后，块大小为 0x0000 向加载程序发出信号，表明传输完成。此时，加载程序将入口点地址返回给调用例程，调用例程将依次清理并退出。然后，执行将在由输入数据流内容确定的入口点地址继续。在 8 位模式下，首先发送数据的最低有效字节（LSB），然后发送最高有效字节（MSB）。对于 32 位值（例如目标地址），首先加载最高有效字（MSW），然后加载最低有效字（LSW）。引导加载程序在加载 8 位数据流时会考虑到这一点。

表格 21 数据流中的 LSB/MSB 加载序列

字节		LSB	MSB
1	2	LSB:AA（内存宽度的键值=8 位）	MSB:08h（内存宽度的键值=8 位）
3	4	LSB:寄存器初始化值或保留值	MSB:寄存器初始化值或保留值
5	6	LSB:寄存器初始化值或保留值	MSB:寄存器初始化值或保留值
7	8	LSB:寄存器初始化值或保留值	MSB:寄存器初始化值或保留值
…	…	…	…
…	…	…	…
17	18	LSB:寄存器初始化值或保留值	MSB:寄存器初始化值或保留值
19	20	LSB:入口点 PC 的上半部分[23:16]	MSB:入口点 PC 的上半部分[31:24]
21	22	LSB:入口点 PC 的下半部分[7:0]	MSB:入口点 PC 的下半部分[15:8]
23	24	LSB:以第一个要加载的块为单位的块大小（以两个字节为单位）。如果块大小为 0，则表明源程序已结束。否则，接下来的另一个块。例如，大小为 0x000A 将指示该块中的 20 个字节。	MSB:块大小
25	26	LSB:MSW 目标地址，第一块地址[23:16]	MSB:MSW 目标地址，第一个块地址[31:24]
27	28	LSB:LSW 目标地址，第一个块 Addr[7:0]	MSB:LSW 目标地址，第一个块地址[15:8]
29	30	LSB:正在加载的第一个块的第一个字	MSB:正在加载的第一个块的第一个字
…	…	…	…
…	…	…	…
		LSB:要加载的第一个块的最后一个字	MSB:正在加载的第一个块的最一个字
		LSB:第二个块的块大小	MSB:第二个块的块大小

字节		LSB	MSB
		LSB:MSW 目标地址，第二块地址[23:16]	MSB:MSW 目标地址，第二个块地址[31:24]
		LSB:LSW 目标地址，第二个块 Addr[7:0]	MSB:LSW 目标地址，第二个块地址[15:8]
		LSB:正在加载的第二个块的第一个字	MSB:正在加载的第二个块的第一个字
…	…	…	…
…	…	…	…
		LSB:第二个块的最后一个字	MSB:第二个块的最后一个字
		LSB:最后一个块的块大小	MSB:最后一个块的块大小
		LSB:MSW 目标地址，最后一块地址[23:16]	MSB:MSW 目标地址，最后一块地址[31:24]
		LSB:LSW 目标地址，最后一块地址 Addr[7:0]	MSB:LSW 目标地址，最后一块地址[15:8]
		LSB:正在加载最后一个块的第一个字	MSB:正在加载最后一个块的第一个字
…	…	…	…
…	…	…	…
n	n+1	LSB:最后一个块的最后一个字	MSB:最后一个块的最后一个字

表格 22 数据流中的 LSB/MSB 加载序列实际示例

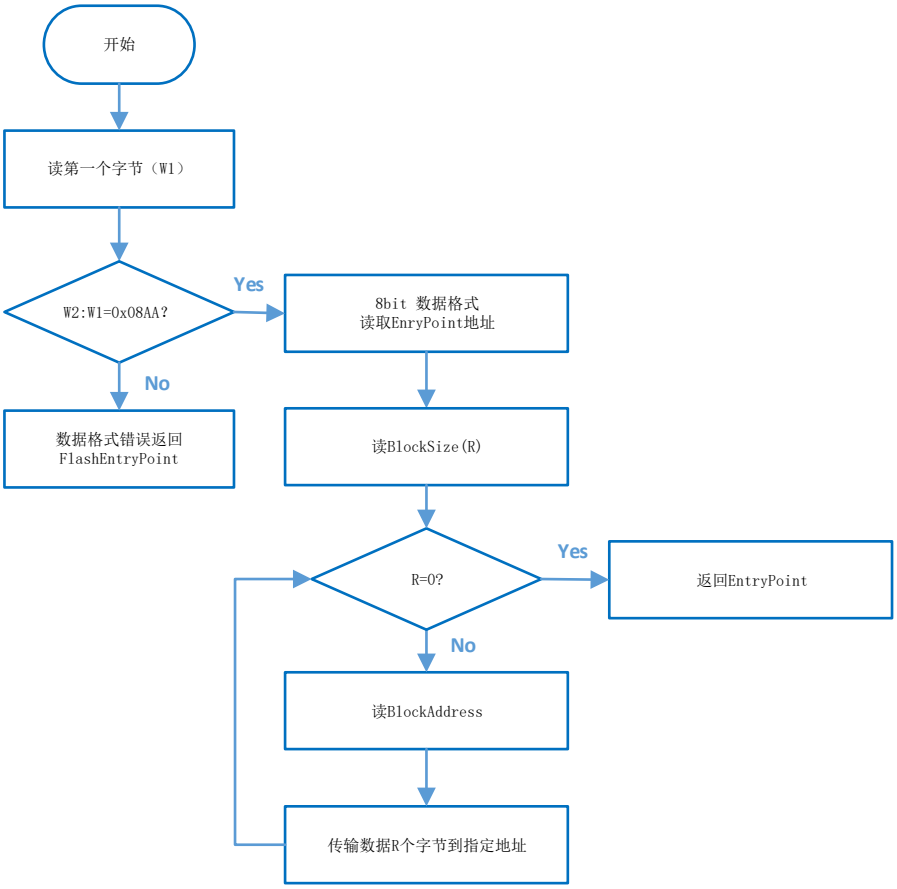
LSB	MSB	LSB	MSB	描述
AA	08	;	;	0x08AA 8 位键值
00	00	00	00	8 个保留字
00	00	00	00	
00	00	00	00	
00	00	00	00	
10	20	00	00	0x20100000 SRAM EntryAddr，启动加载完成后的起点
06	00	;	;	0x0006-第一个块由 12 个字节组成
10	20	00	00	0x20100000 -第一个块将从 0x20100000 开始加载
01	00	;	;	数据加载 = 0x0001 0x0002 0x0003 0x0004 0x0005 0x0006
02	00	;	;	
03	00	;	;	
04	00	;	;	
05	00	;	;	
06	00			
02	00	;	;	0x0002-第二个块由 4 个字节组成
10	20	0C	00	0x2010000C-第二块将从 0x201000C 开始加载
34	12	;	;	数据加载 = 0x1234 0x5678
78	56	;	;	
00	00	;	;	0x0000-大小为 0 表示数据流结束

4.5. 基本传输步骤

图 2 阐述了 Bootloader 加载数据流、传输该数据以及开始执行程序的基本进程。数据流结构中

的所有值均采用十六进制。当 bootloader 找到根据 GPIO 引脚的状态选定的有效引导模式之后就会开始执行此进程。加载程序首先将主机发送的第一个 16bit 与键值 0x08AA 进行比较。如果加载程序发现此报头与该键值不匹配，或者如果该值对给定的引导模式无效，加载则中止。返回至闪存的起始地址。

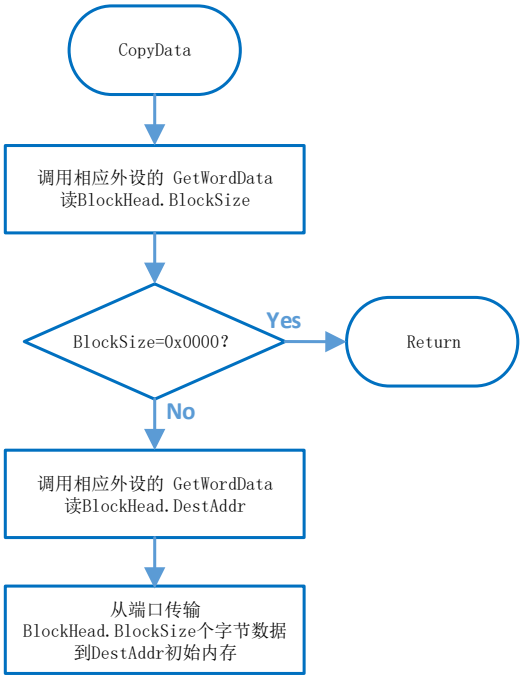
图 2 数据传输流程



4.6. CopyData 函数

每个 Bootloader 都使用相同的函数将数据从端口复制至 RAM。此函数就是 CopyData()函数。它使用一个指针指向 GetWordData 函数，后者由每个加载程序初始化以正确从该端口读取数据。例如，在调用 UART 加载程序时，GetWordData 函数指针被初始化，以指向特定于 UART 的 UART_GetWordData 函数，以便在调用 CopyData()函数时可访问正确的端口。CopyData 函数的流程如图 3 所示：

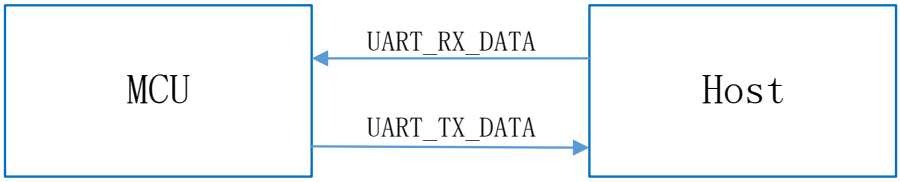
图 3 CopyData 执行流程



4.7. UART_Boot 函数

UART 引导模式可以将代码从 UART-A 异步传输至内存储器。此引导模式仅支持传入的 8 位数据流，以及遵循 4.4 章节的相同数据流。

图 4 UART Boot 操作概述



MCU 通过 UART-A 外设与外部主机器件通信。UART 端口的自动波特特性用于锁定与主机通信的波特率。因此，UART loader 非常灵活，您可以使用多个不同的波特率与 MCU 通信。

在每个数据传输之后，MCU 会将收到的 8 位字符回波给主机。通过这种方式，主机可以检查 MCU 是否收到了每个字符。

如果波特率较高，传入数据位的转换率则受收发器和连接器的性能影响。虽然常规串行通信可以运作良好，但此转换率可能会限制在较高波特率（通常高于 100k 波特）时执行可靠的自动波特检测，并导致自动波特锁定特性失效。为避免出现这种情况，建议执行以下操作：

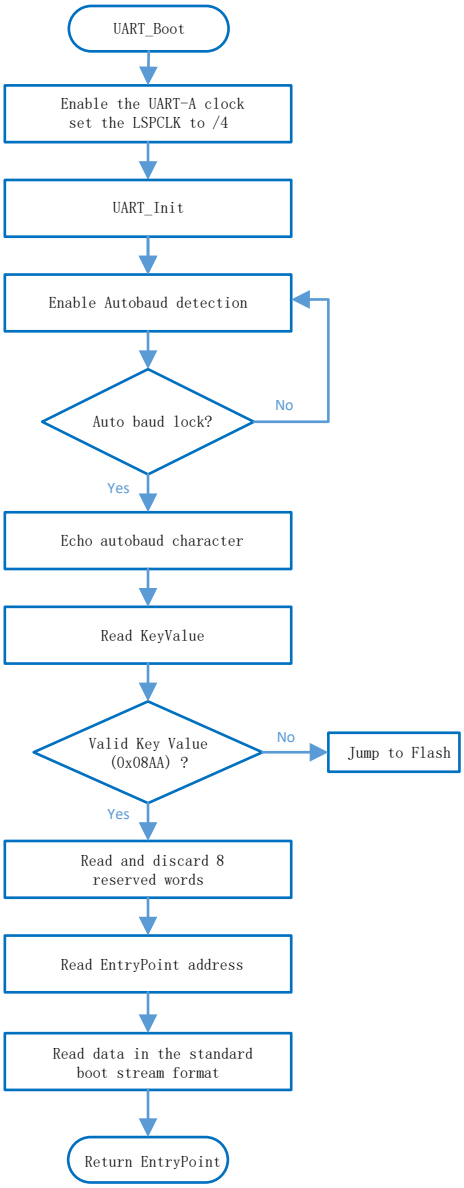
（1） 使用较低的波特率实现主机与 UART bootloader 之间的波特锁定。

（2） 在此较低的波特率下加载传入的应用程序或定制的加载程序。

（3） 主机可以与加载的应用程序握手，将 UART 波特率寄存器设置为所需的高波特率。

UART Boot 函数的流程如图 5 所示：

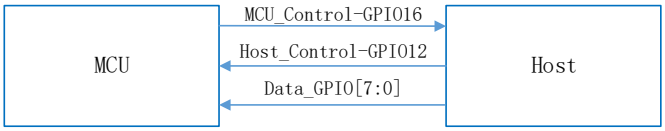
图 5 UART Boot 执行流程



4.8. Parallel_Boot 函数

并行通用 I/O（GPIO）引导模式可以将代码从 GPIO0-GPIO7 异步传输至内存储器。每一个值的长度都可以为 8 位，以及遵循 4.4 章节的相同数据流。

图 6 Parallel Boot 操作概述



并行 GPIO 加载程序使用以下引脚:

- GPIO 数据[7:0]。
- GPIO16 上的 MCU 系列控制。
- GPIO12 上的主机控制。

R501 系列通过轮询/驱动 GPIO12 和 GPIO16 线路与外部主机设备通信。GPIO12 需要一个外部上拉电阻，因为 GPIO 引脚缺少防止 R501 系列过早读取数据所需的内部上拉电路。根据您的系统，GPIO16 可能还需要外部上拉。使用图 7 中所示的握手协议通过 GPIO 成功传输每个字节[7:0]。该协议非常强大，允许较慢或较快的主机与 R501 系列通信。

读取两个连续的 8 位字以形成单个 16 位字。首先读取最高有效字节（MSB），然后读取最低有效字节（LSB）。在这种情况下，从 GPIO[7:0]读取数据。8 位数据流如表格 23 所示。

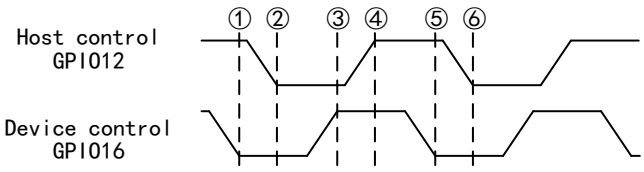
表格 23 并行 GPIO 引导 8 位数据流

字节		LSB	MSB	描述
1	2	AA	08	0x08AA 8 位键值
3	4	00	00	8 个保留字
…	…	…	…	…
17	18	00	00	最后的保留字
19	20	BB	AA	入口点 PC[31:16]
21	22	DD	CC	入口点 PC[15:0]（PC=0xAABBCCDDD）
23	24	NN	MM	第一个加载的数据块大小 =0xMMNN 字节组成
25	26	BB	AA	第一个块地址的目的地址[31:16]
27	28	DD	CC	第一个块地址的目的地址[15:0]（地址=0xAABBCCDD）
29	30	FF	EE	加载第一个块的第一个数据 =0xEEFF
…	…	…	…	…
…	…	…	…	该段数据
…	…	…	…	…
		ZZ	YY	加载第一个块的最后一个数据 =0xYYZZ
		NN	MM	第一个加载的数据块大小 =0xMMNN 字节组成
		BB	AA	第二个块地址的目的地址[31:16]
		DD	CC	第二个块地址的目的地址[15:0]（地址=0xAABBCCDD）
		FF	EE	加载第二个块的第一个数据 =0xEEFF
		…	…	
N	N+1	ZZ	YY	加载最后一个块的最后一个数据 =0xYYZZ
N+2	N+3	00	00	块大小为 0000h 表示数据传输完成。

R501 系列设备首先通过将 GPIO16 引脚拉低来向主机发出信号，表明它已准备好开始数据传输。然后，主机加载通过将 GPIO12 引脚拉低来启动数据传输。

并行 GPIO 引导加载程序握手协议如图 7：

图 7 Parallel BootLoader 握手协议



R501 系列设备表示已准备好通过将 GPIO16 引脚拉低来开始接收数据。

（1） 引导加载程序等待，直到主机将数据放在 GPIO 上[7:0]。通过将 GPIO12 引脚拉低，主机向 R501 系列设备发出信号，表明数据已准备就绪。

（2） R501 设备读取数据，并通过将 GPIO16 拉高向主机发出读取完成的信号。

（3） 引导加载程序通过将 GPIO12 拉高直到主机确认 R501 系列。

（4） R501 设备再次通过将 GPIO6 引脚拉低表示它已准备好接收更多数据。对于要发送的每个数据值，重复此过程。

图 8 Parallel Boot 执行流程

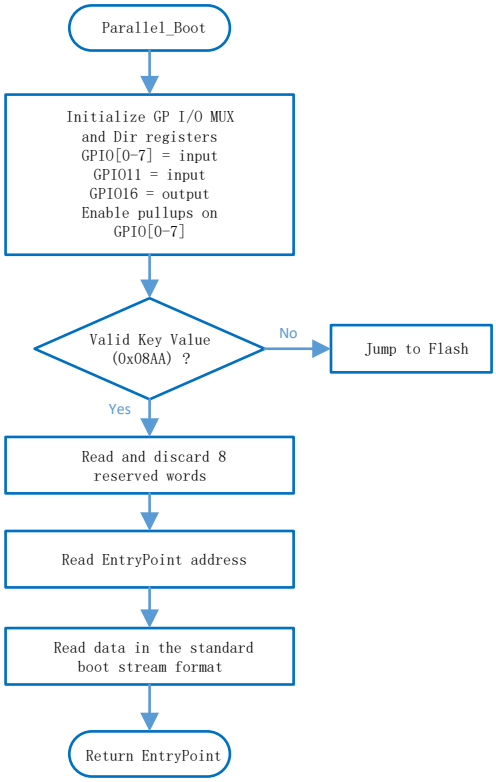
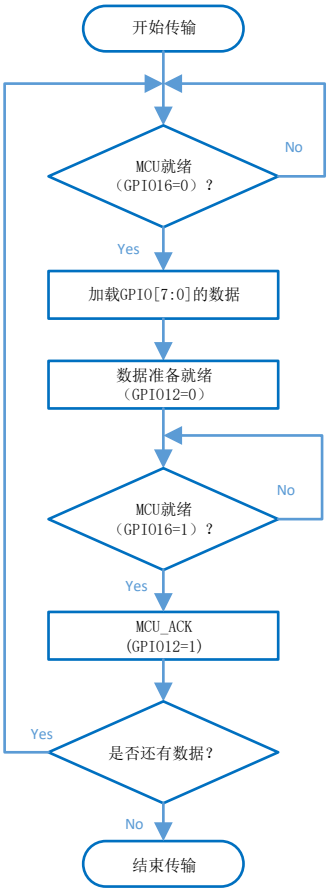


图 9 显示了从主机端传输的流程。在此模式下，CPU 和主机的运行速度并不关键，因为主机将等待 R501 系列，而 R501 也会等待主机。通过这种方式，该协议将同时运行比 R501 系列更快的主机和运行更慢的主机。

图 9 Parallel Boot 主机执行流程



4.9. SPI_Boot 函数

SPI 引导 ROM 加载器初始化 SPI 模块以与 SPI 兼容、16 位或 24 位可寻址串行 EEPROM 或闪存设备接口。如图 10 所示，期望这种器件存在于 SPI-A 引脚上。SPI 引导加载程序支持 8 位数据流。

图 10 SPI Boot 操作概述

SPI-A 加载程序使用以下引脚:

- GPIO16 上的 SPIA_SIMO
- GPIO17 上的 SPIA_SOMI
- GPIO18 上的 SPIA_CLKA

- GPIO19 上的 SPIA_ESTE_

SPI 引导只读存储器加载器通过以下设置初始化 SPI:启用先进先出、8 位字符、内部 SPICLK 主模式和通话模式、时钟相位等于 1、极性等于 0、初始传输速度设置为最低，以确保设备能够顺利接收数据。

如果要从另一个设备上的 SPI 端口执行下载，则必须将该设备设置为在从属模式下运行并模拟串行 SPI EEPROM。进入 SPI_Boot 功能后，SPI 引脚的引脚功能立即设置为主引脚，并初始化 SPI。初始化以尽可能慢的速度完成。一旦 SPI 初始化并读取键值，您可以指定波特率或低速外围时钟的变化。

表格 24 SPI 8 位数据流

字节	描述
1	LSB：AA
2	MSB：08h
3	LSB：LOSPCP
4	MSB：SPIBRR
5	LSB：保留未使用
6	MSB：保留未使用
…	…
…	...
17	LSB：保留未使用
18	MSB：保留未使用
19	LSB：入口点 PC 的上半部分[23:16]
20	MSB：入口点 PC 的上半部分[31:24]
21	LSB：入口点 PC 的下半部分[7:0]
22	MSB：入口点 PC 的下半部分[15:8]
…	…
…	…
…	如通用数据流说明中所示的“大小/目的地址/数据”格式的数据块
…	…
…	…
N	LSB：00h
N+1	MSB：00h 表示结束

传输完全以字节模式（8 位/字符的 SPI）执行。下面列出了此顺序的逐步说明：

（1） 初始化 SPI-A 端口

（2） 将 GPIO19（SPISTE）引脚用作串行 SPI EEPROM 的片选。

（3） SPI-A 为串行 SPI EEPROM 输出一个读取命令。

（4） SPI-A 向串行 SPI EEPROM 发送地址 0x0000；即，主机要求 EEPROM 的可下载数据

包必须从 EEPROM 中的地址 0x0000 处开始。

（5） 获取的下个字必须与 8 位数据流（0x08AA）的键值相匹配。

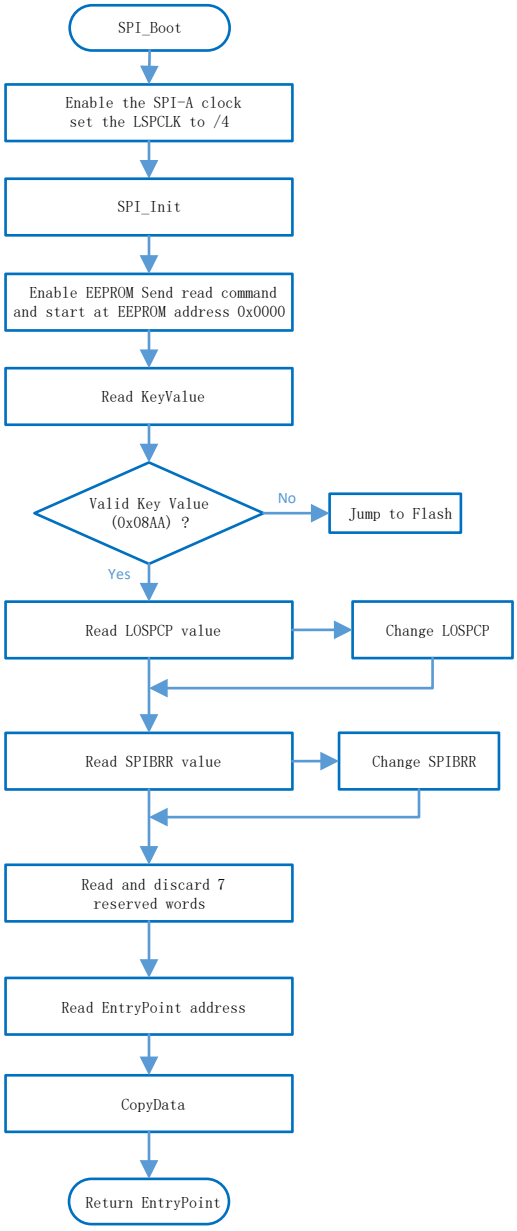
（6） 此字的最低有效字节是首次读取的那个字节，而最高有效字节是获取的下一个字节。在 SPI 中所有字传输都为真。如果键值不匹配，加载则中止，并将闪存的应用起点（0x08000000）返回至调用例程。

（7） 接下来获取的两个字节可用于更改低速外设时钟寄存器（LOSPCP）和 SPI 波特率寄存器（SPIBRR）的值。读取的第一个字节是 LOSPCP 值，第二个字节是 SPIBRR 值。SPI bootloader 将读取这 14 个字节，然后丢弃。

（8） 接下来的 4 个字节构成了 32 位应用起点地址，在完成引导加载程序之后，将继续从该地址执行程序；它通常是通过 SPI 端口下载的程序的应用起点。

（9） 通过 SPI 端口将多个代码块和数据块从外部串行 SPI EEPROM 复制到存储器中。代码块按之前介绍的标准数据流结构组织在一起。这一操作直至遇到 0x0000 块大小时才完成。此时将应用起点地址返回至调用例程，后者然后退出 bootloader，并在所指定的地址处继续执行。

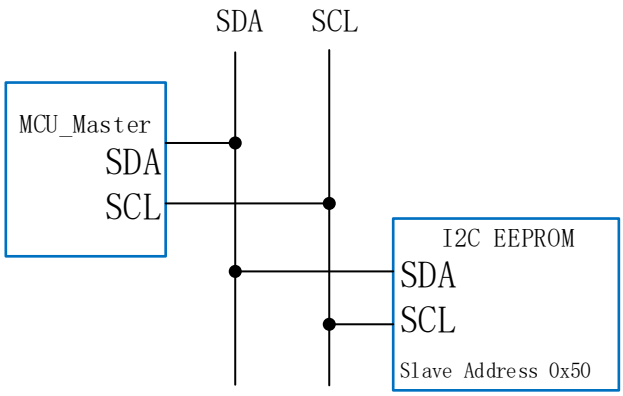
图 11 SPI Boot 执行流程



4.10. I2C_Boot 函数

I2C bootloader 预计在 I2C-A 总线上的 0x50 地址处存在的是一个 8 位宽的与 I2C 兼容的 EEPROM 器件。此 EEPROM 必须遵守采用 16 位基址结构的传统的 I2C EEPROM 协议（如此部分所述）。

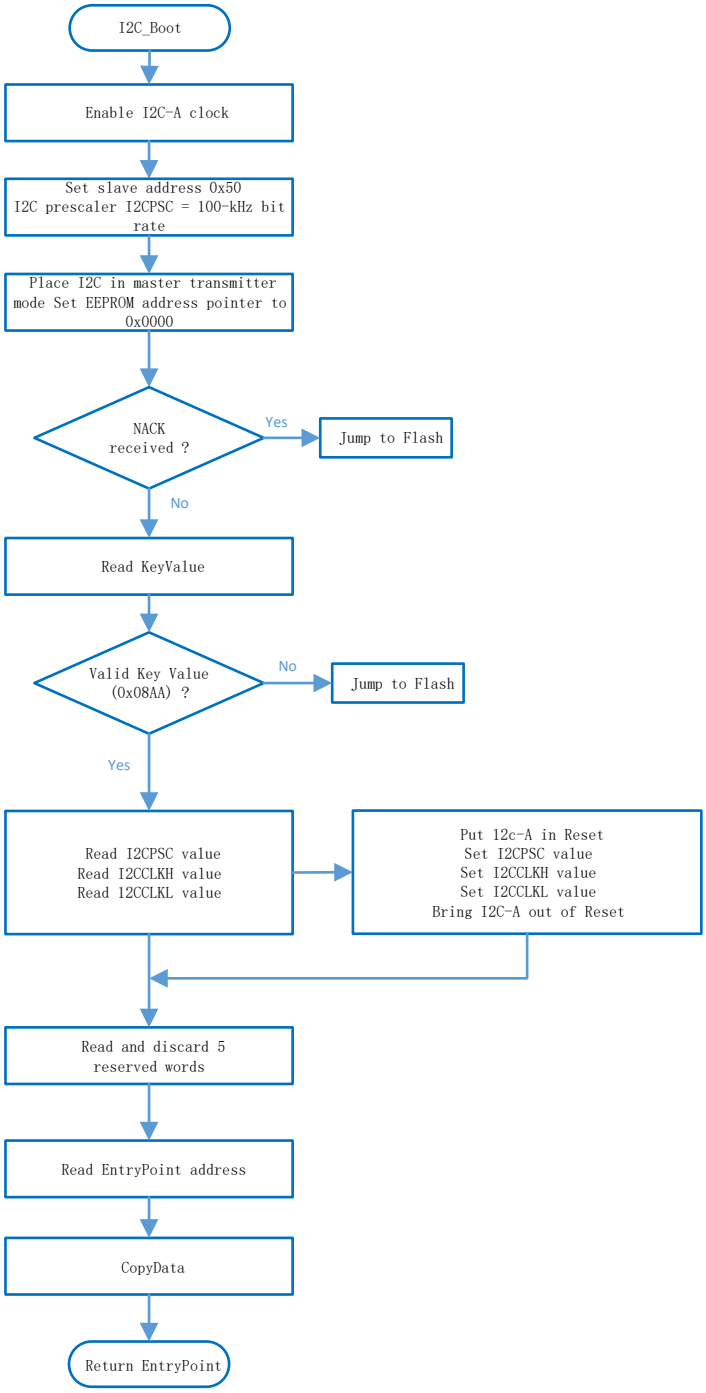
图 12 I2C Boot 操作概述



如果要从非 EEPROM 器件执行下载，则必须将该器件设置为在从模式下操作并模拟 I2C EEPROM。输入 I2C Boot 函数之后，立即针对 I2C-A 操作配置 GPIO 引脚并初始化 I2C。从 I2C 模块引导时，必须满足以下要求：

- 设备的输入频率必须在适当的范围内。
- EEPROM 必须在从机地址 0x50 处。

图 13 I2C Boot 执行流程



要使用 I2C-A bootloader，默认情况下，引导加载器将预分频器（I2CCLKH 和 I2CCLKL）配置为在系统时钟为 10 MHz 时，以 100 kHz 的比特率（标准 I2C 模式）运行 I2C。配置为 50%的占空比，以确保稳定的时钟信号。在从 EEPROM 接收到前几个字节后，可以修改这些寄存器，以将通信速率提高到 400 kHz（快速 I2C 模式），用于后续的数据读取。

启动期间不检查仲裁、总线忙碌和从设备信号。因此，在初始化阶段不允许其他主设备控制总线。如果应用程序需要在 I2C 启动模式下使用另一个主设备，则必须配置该主设备，以确保在应用软件信号表明启动加载器初始化完成之前，不发送任何 I2C 消息。

非确认位仅在发送初始化 EEPROM 基地址的第一条消息时检查，以确认地址 0x50 处是否存在 EEPROM。

如果 EEPROM 不存在，则在数据读取消息的地址阶段（I2C_Get Word）不检查非确认位。

如果在数据读取消息期间收到非确认响应，I2C 总线将挂起。表格 25 显示了 I2C 使用的 8 位数据流：

表格 25 I2C 8 位数据流

字节	描述
1	LSB：AA
2	MSB：08h
3	LSB：I2CPSC
4	保留
5	LSB：I2CCLKH[7:0]
6	MSB：I2CCLKH[15:8]
7	LSB：I2CCLKL[7:0]
8	MSB：I2CCLKL[15:8]
…	…
…	...
17	LSB：保留未使用
18	MSB：保留未使用
19	LSB：入口点 PC 的上半部分[23:16]
20	MSB：入口点 PC 的上半部分[31:24]
21	LSB：入口点 PC 的下半部分[7:0]
22	MSB：入口点 PC 的下半部分[15:8]
…	…
…	…
…	如通用数据流说明中所示的“大小/目的地址/数据”格式的数据块
…	…
…	…
N	LSB：00h
N+1	MSB：00h 表示结束

I2C bootloader 所需的 I2C EEPROM 协议如图 14 与图 15。第一次通信（用于将 EEPROM 地址指针指向 0x0000 并从该处读取键值（0x08AA））如图 14 所示。 所有后续读取如图 15 所示，并且一次读取两个字节。

图 14 随机读取

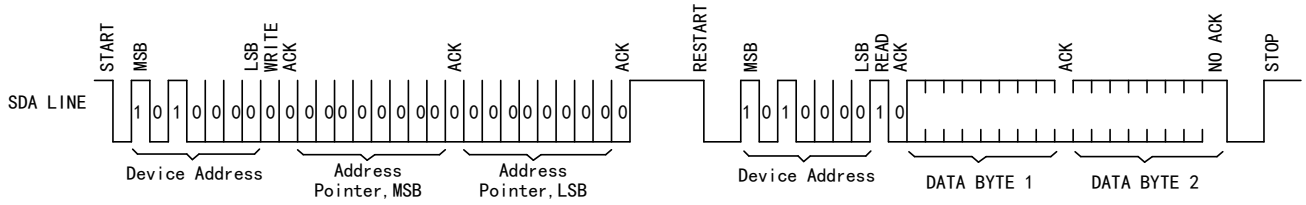
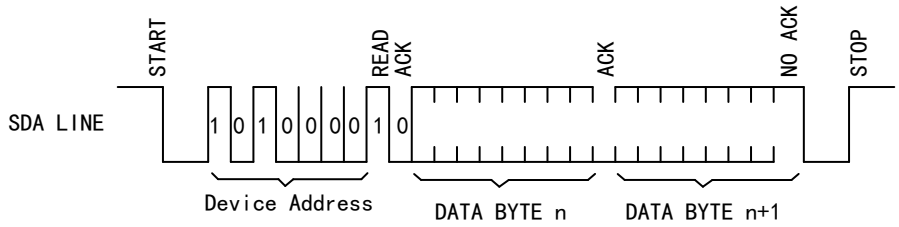


图 15 顺序读取



4.11. CAN_Boot 函数

CAN bootloader 将代码从 CAN-A 异步传输至内存储器。主机可以是任意 CAN 节点。首先使用 11 位标准标识（具有一个等于 0x1 的 MSGID）按每数据帧两个字节完成通信。如果需要更大的数据吞吐量，主机可以下载一个内核来重新配置 CAN。

图 16 CAN Boot 操作概述

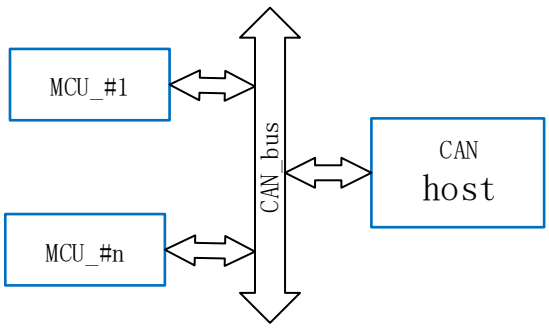
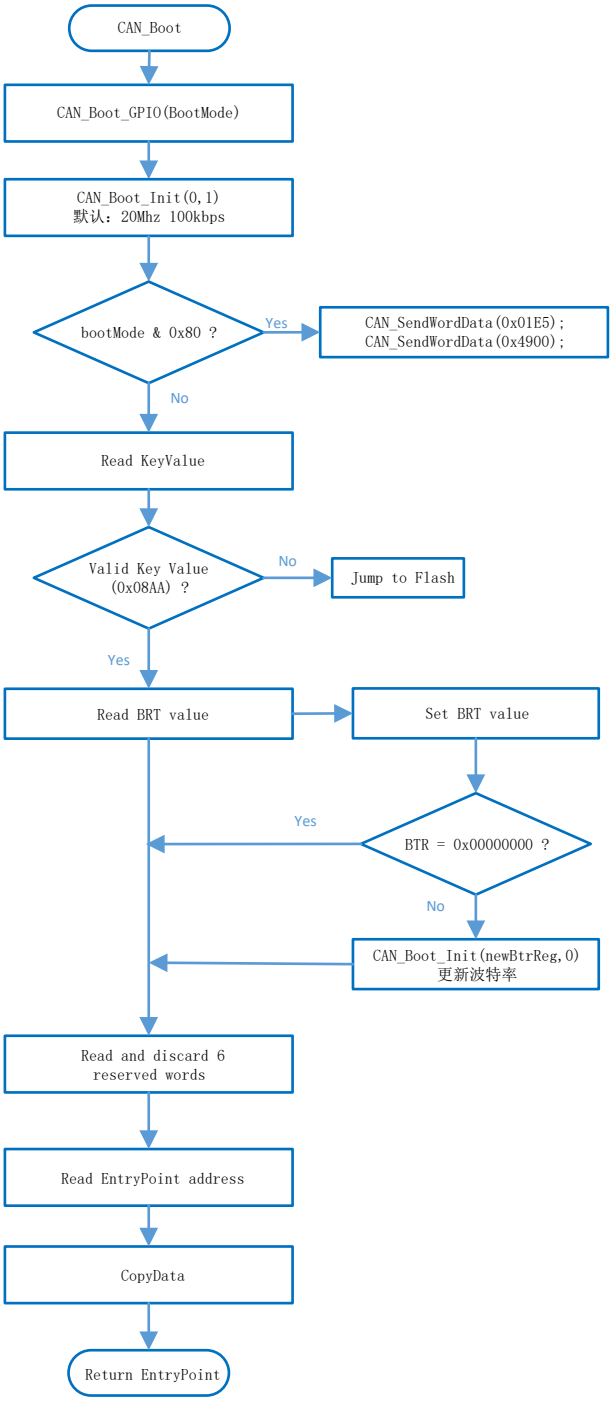


图 17 CAN Boot 执行流程



所示的 SYSCLKOUT 值是采用默认 PLL 设置时的复位值。BRPreg 和位时序值已被分别硬编码成 10 和 20。邮箱 1 使用等于 0x1 的标准 MSGID 进行编程，用于 boot-loader 通信。CAN 主机应当一次只发送 2 个字节，首先发送 LSB，然后是 MSB。例如，要将字 0x08AA 发送至 R501，则首先发送 AA，然后是 08。CAN bootloader 的程序流程与 UART bootloader 相同。CAN bootloader 的数据顺序如表格 26 所示

表格 26 CAN 8 位数据流

字节		LSB	MSB	描述
1	2	AA	08	0x08AA 8 位键值
3	4	00	00	8 个保留字
…	…	…	…	…
17	18	00	00	最后的保留字
19	20	BB	AA	入口点 PC[31:16]
21	22	DD	CC	入口点 PC[15:0]（PC=0xAABBCCDDD）
23	24	NN	MM	第一个加载的数据块大小 =0xMMNN 字节组成
25	26	BB	AA	第一个块地址的目的地址[31:16]
27	28	DD	CC	第一个块地址的目的地址[15:0]（地址=0xAABBCCDD）
29	30	FF	EE	加载第一个块的第一个数据 =0xEEFF
…	…	…	…	…
…	…	…	…	…
…	…	…	…	…
		ZZ	YY	加载第一个块的最后一个数据 =0xYYZZ
		NN	MM	第一个加载的数据块大小 =0xMMNN 字节组成
		BB	AA	第二个块地址的目的地址[31:16]
		DD	CC	第二个块地址的目的地址[15:0]（地址=0xAABBCCDD）
		FF	EE	加载第二个块的第一个数据 =0xEEFF
		…	…	
N	N+1	ZZ	YY	加载最后一个块的最后一个数据 =0xYYZZ
N+2	N+3	00	00	块大小为 0000h 表示数据传输完成。

CAN 引导加载程序提供了一个选项，可以增加默认的比特率。默认情况下，CAN 位定时寄存器（CAN_BTR）被编程为在 20MHz 外部振荡器下实现 100 kbps 的比特率。如果希望更快的比特率，主机应该在比特流中提供某些参数。这些参数将用于修改 CANBTR 寄存器，从而改变比特率。CAN_BTR 决定当前的比特率，以 32 位值表示。

在主机侧生成的应用程序十六进制文件中，8 位数据流具有以下字节序列：AA,08,00,00,...,00（16 个保留字节）。KeyValue 是前两个字节（0x08AA），以 LSB-MSB 格式排序。通过使用主机编程器，在引导加载程序识别 KeyValue 并解析保留字后，可以使用新的（非零）位定时寄存器值来更改 CAN_BTR 寄存器的值。例如，假设一个应用程序想将比特率提高到 1 Mbps。将值 0x7AC0 写入 CAN_BTR 可以实现这一点（请注意，给定的比特率可以通过不同的 TSEG1 和 TSEG2 值组合来实现；0x7AC0 仅作为一个示例）。为了选择新的比特率（从 100 kbps 到 1 Mbps），生成的应用程序文本文件需要修改为：AA,08,C0,7A,…,00。此外，主机编程器还应修改其自身的比特率，以便在新的比特率下继续与引导加载程序通信。

4.12. Secure_Boot 函数

安全闪存引导模式类似于闪存引导模式，不同之处在于引导流程在闪存内容经过认证后才会分支

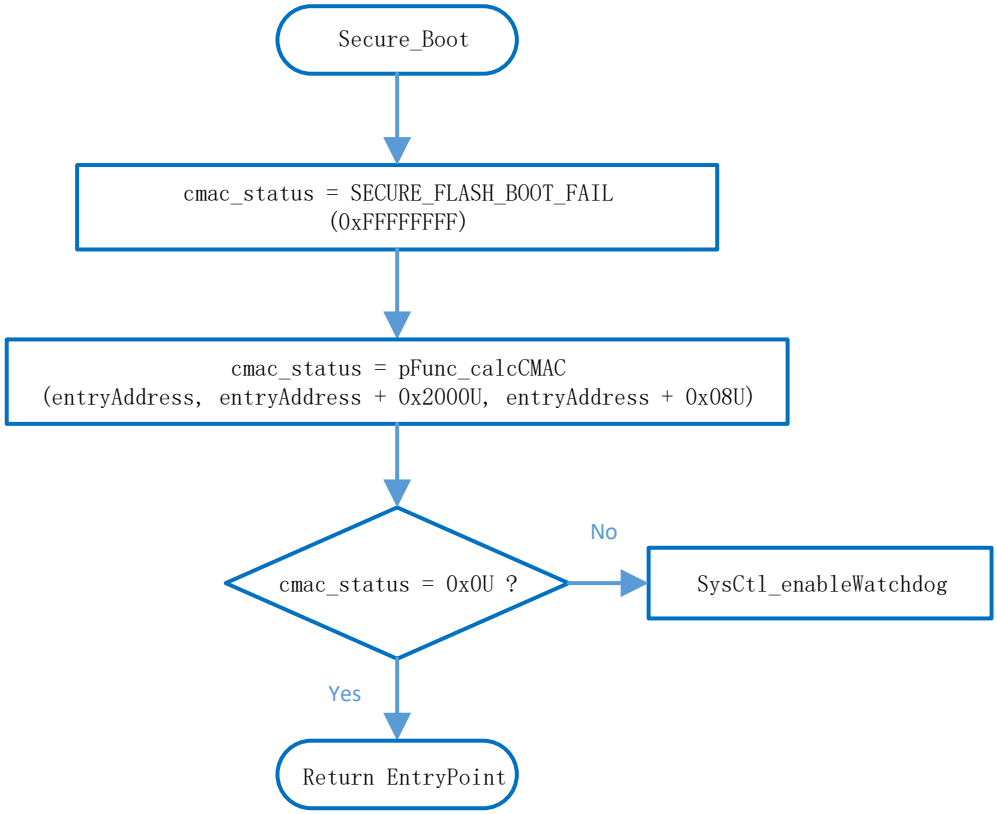
到配置的闪存地址。闪存认证使用基于密码的消息认证协议（CMAC）来认证从配置的闪存入口点地址开始的 8KB 闪存。CMAC 计算需要一个用户定义的 128 位密钥，该密钥被编程在 CPU1 用户 OTP 区的 OTP CMACKEY 位字段中。此外，用户必须根据 8KB 闪存范围计算 golden CMAC 标签，并将其与用户代码一起存储在闪存中的硬编码地址。

在安全闪存引导过程中，计算出的 CMAC 标签与闪存中的用户 golden CMAC 标签进行比较，以确定 CMAC 认证的通过/失败状态。当认证通过时，引导流程继续并分支到闪存以开始执行应用程序。当认证失败时，会进入 WaitBoot。

详细 Secure Boot 的配置和使用可参考 AN1139 应用笔记。

有关可用的安全闪存引导入口地址选项，请参阅第 3.2 节。

图 18 Secure Boot 执行流程



5. 版本历史

表格 27 文件版本历史

日期	版本	变更历史
2025.01	1.0	新建

声明

本手册由珠海极海半导体有限公司（以下简称“极海”）制订并发布，所列内容均受商标、著作权、软件著作权相关法律法规保护，极海保留随时更正、修改本手册的权利。使用极海产品前请仔细阅读本手册，一旦使用产品则表明您（以下称“用户”）已知悉并接受本手册的所有内容。用户必须按照相关法律法规和本手册的要求使用极海产品。

1、权利所有

本手册仅应当被用于与极海所提供的对应型号的芯片产品、软件产品搭配使用，未经极海许可，任何单位或个人均不得以任何理由或方式对本手册的全部或部分内容进行复制、抄录、修改、编辑或传播。

本手册中所列带有“®”或“™”的“极海”或“Geehy”字样或图形均为极海的商标，其他在极海产品上显示的产品或服务名称均为其各自所有者的财产。

2、无知识产权许可

极海拥有本手册所涉及的全部权利、所有权及知识产权。

极海不应因销售、分发极海产品及本手册而被视为将任何知识产权的许可或权利明示或默示地授予用户。

如果本手册中涉及任何第三方的产品、服务或知识产权，不应被视为极海授权用户使用前述第三方产品、服务或知识产权，也不应被视为极海对第三方产品、服务或知识产权提供任何形式的保证，包括但不限于任何第三方知识产权的非侵权保证，除非极海在销售订单或销售合同中另有约定。

3、版本更新

用户在下单购买极海产品时可获取相应产品的最新版的手册。

如果本手册中所述的内容与极海产品不一致的，应以极海销售订单或销售合同中的约定为

准。

4、信息可靠性

本手册相关数据经极海实验室或合作的第三方测试机构批量测试获得，但本手册相关数据难免会出现校正笔误或因测试环境差异所导致的误差，因此用户应当理解，极海对本手册中可能出现的该等错误无需承担任何责任。本手册相关数据仅用于指导用户作为性能参数参照，不构成极海对任何产品性能方面的保证。

用户应根据自身需求选择合适的极海产品，并对极海产品的应用适用性进行有效验证和测试，以确认极海产品满足用户自身的需求、相应标准、安全或其它可靠性要求；若因用户未充分对极海产品进行有效验证和测试而致使用户损失的，极海不承担任何责任。

5、合规要求

用户在使用本手册及所搭配的极海产品时，应遵守当地所适用的所有法律法规。用户应了解产品可能受到产品供应商、极海、极海经销商及用户所在地等各国有关出口、再出口或其它法律的限制，用户（代表其本身、子公司及关联企业）应同意并保证遵守所有关于取得极海产品及/或技术与直接产品的出口和再出口适用法律与法规。

6、免责声明

本手册由极海“按原样”（as is）提供，在适用法律所允许的范围内，极海不提供任何形式的明示或暗示担保，包括但不限于对产品适销性和特定用途适用性的担保。

极海产品并非设计、授权或担保适合用于军事、生命保障系统、污染控制或有害物质管理系统中的关键部件，亦非设计、授权或担保适合用于在产品失效或故障时可导致人员受伤、死亡、财产或环境损害的应用。

如果产品未标明“汽车级”，则表示不适用于汽车应用。如果用户对产品的应用超出极海提供的规格、应用领域、规范，极海不承担任何责任。

用户应该确保对产品的应用符合相应标准以及功能安全、信息安全、环境标准等要求。用户对极海产品的选择和使用负全部的责任。对于用户后续在针对极海产品进行设计、使用的过程中

所引起的任何纠纷，极海概不承担责任。

7、责任限制

在任何情况下，除非适用法律要求或书面同意，否则极海和/或以“按原样”形式提供本手册及产品的任何第三方均不承担损害赔偿责任，包括任何一般、特殊因使用或无法使用本手册及产品而产生的直接、间接或附带损害（包括但不限于数据丢失或数据不准确，或用户或第三方遭受的损失），这涵盖了可能导致的人身安全、财产或环境损害等情况，对于这些损害极海概不承担责任。

8、适用范围

本手册的信息用以取代本手册所有早期版本所提供的信息。