摘 要

本应用笔记旨在帮助指导用户针对芯海 CORTEX-M3 MCU CS32F103 系列单片机 IAP 应用的快速开发。本应用笔记实现了 CAN 和 USART 两种接口方式来开发 IAP 应用，协议层使用的是芯海内部协议，仅供参考。

适用范围

类型	适用产品型号或系列	说明
MCU	所有 ARM CORTEX-M3 内核的 MCU，包括：CS32F103 系列	

版 本

历史版本	修改内容	日期
V1.0	初版生成	2022-11-30

目 录

1. IAP 介绍 4
1.1. CS32F103 常规运行流程 5
1.2. CS32F103 IAP 运行流程 6
1.3. 芯海 IAP 运行平台搭建 7
2. CSBOOT IAP 应用指导 8
2.1. 分层结构 8
2.2. 硬件参考设计 9
2.3. 芯片空间分配 9
2.4. CSBOOT 工作流程 9
2.5. CONFIG.H 配置介绍 10
3. CAN 接口注意事项 11
4. 主应用代码编写指导 12
4.1. 向量表偏移 12
4.2. 主程序地址偏移 12
5. CSBOOT 通信协议 13
5.1. 协议帧格式 13
5.2. 协议指令 13
5.2.1. 命令码 13
5.2.2. 响应状态 14
5.2.3. 执行状态 14
5.3. 通信数据 14
5.3.1. Get ID 命令 14
5.3.2. Get Version 命令 15
5.3.3. Write Memory 命令 15
5.3.4. Read Memory 命令 16
5.3.5. Erase Page 命令 16
5.3.6. Erase Chip 命令 17
5.3.7. Go 命令 17
5.3.8. Write Protect 命令 17
5.3.9. Write Unprotect 命令 19
5.3.10. Readout Protect 命令 19
5.3.11. Readout Unprotect 命令 20
5.4. 异常回复 20
6. 上位机应用指导 21
6.1. 上位机地址写入起始地址 21
6.2. IAP 程序延时参数 21
6.3. 上位机操作步骤 21

1. IAP 介绍

IAP，全称是“In-Application Programming”，中文解释为“在程序中编程”。IAP 是一种对通过微控制器的通信接口（如 USART，IIC，CAN，USB，以太网接口甚至是无线射频通道）对正在运行程序的微控制器进行内部程序的更新的技术（注意这完全有别于 ICP 或者 ISP 技术）。

ICP（In-Circuit Programming）技术即通过在线仿真器或烧录器对单片机进行程序烧写，而 ISP 技术则是通过单片机内置的 bootloader 程序引导的烧写技术。无论是 ICP 技术还是 ISP 技术，都需要有机械性的操作如连接下载线，设置跳线帽等。若产品的电路板已经层层密封在外壳中，要对其进行程序更新无疑困难重重，若产品安装于狭窄空间等难以触及的地方，更是一场灾难。但若引入了 IAP 技术，则完全可以避免上述尴尬情况，而且若使用远距离或无线的数据传输方案，甚至可以实现远程编程和无线编程。这绝对是 ICP 或 ISP 技术无法做到的。某种微控制器支持 IAP 技术的首要前提是其必须是基于可重复编程闪存的微控制器。CS32F103 微控制器带有可编程的内置闪存，同时 CS32F103 拥有在数量上和种类上都非常丰富的外设通信接口，因此在 CS32F103 上实现 IAP 技术是完全可行的。

CS32F103 的内部闪存地址起始于 0x8000000，一般情况下，程序文件就从此地址开始写入。此外 CS32F103 是基于 Cortex-M3 内核的微控制器，其内部通过一张“中断向量表”来响应中断，程序启动后，将首先从“中断向量表”取出复位中断向量执行复位中断程序完成启动。而这张“中断向量表”的起始地址是 0x8000004，当中断来临，CS32F103 的内部硬件机制会自动将 PC 指针定位到“中断向量表”处，并根据中断源取出对应的中断向量，并赋给 PC 指针，执行中断服务程序。最后还需要知道关键的一点，通过修改 CS32F103 工程的链接脚本可以修改程序文件写入闪存的起始地址。

1.1. CS32F103 常规运行流程

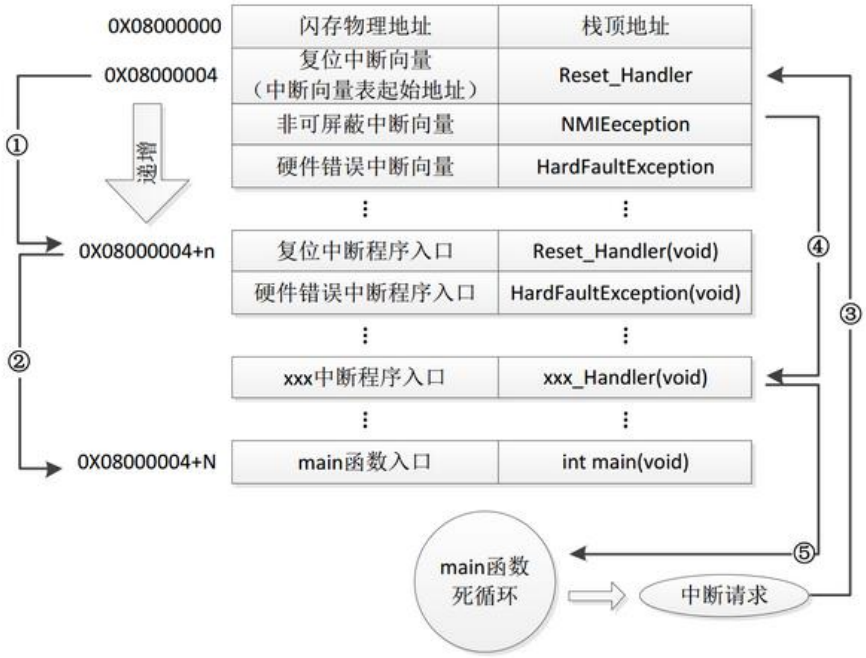

图 1　常规运行流程

如上图所示，芯片是有一套固定的运行流程，

① CS32F103 reset 之后，硬件固定从 0x8000004 地址处取出复位中断向量的地址，并执行复位中断服务程序 Reset_Handler。

② Reset_Handler 在启动文件中有定义，执行 SystemInit 函数后，跳转进入 main 函数。

③ main 函数中发生了中断请求，芯片会将 PC 指针强制指回中断向量表处。

④ 根据中断向量表跳转到中断服务函数。

⑤ 中断程序处理完毕，返回到中断前 main 中的位置。

1.2. CS32F103 IAP 运行流程

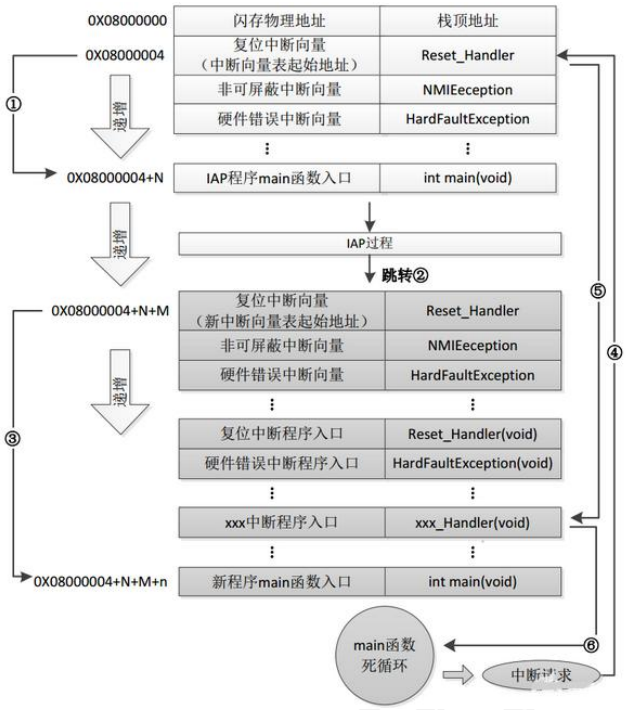

图 2　IAP 运行流程

加入 IAP 程序后，芯片运行流程如上图所示，解读如下：

① CS32F103 reset 之后，硬件固定从 0x8000004 地址处取出复位中断向量的地址，并执行复位中断服务程序 Reset_Handler。随后进入 IAP 程序的 main 函数。

② 执行完 IAP 程序后，跳转至新程序的复位向量表(即图中 0x8000004+N+M 地址)。

③ 随后执行新程序的 main 函数。新程序的 main 函数也具有永不返回的特性，注意，此时 FLASH 上出现了两个中断向量表。

④ 新程序 main 函数执行中，中断请求发生，PC 指针会转向 0x8000004 中断向量表处，而不是新程序的中断向量表，这是由硬件机制决定的。

⑤ 程序会根据我们设定的中断向量表偏移量，跳转到对应的中断服务函数，此时是跳转到新程序的中断服务程序中。

⑥ 中断程序处理完毕，返回到中断前 main 中的位置。

1.3. 芯海 IAP 运行平台搭建

本应用笔记开发和测试涉及的软硬件平台、工具如下。

硬件：CS32F103_LQFP48 开发板 V1.0

调试工具：J-LINK 调试工具、CH340 转串口、USART 转 CAN 收发器

软件：MDK(IDE 开发工具)、CS32 ISP ProgrammerV1.0.3(IAP 下载工具)

IAP 硬件环境如下图所示：

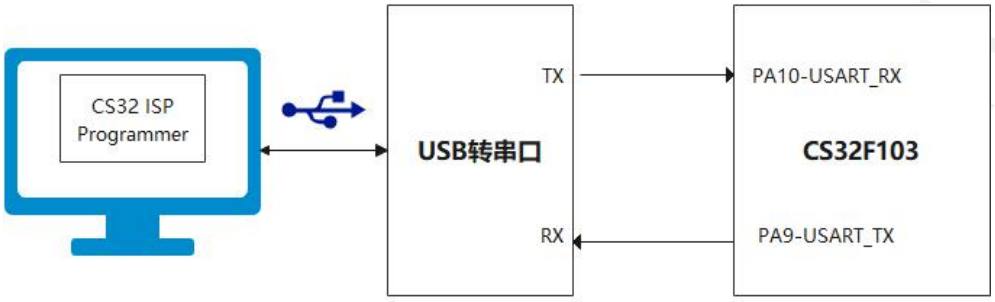

图 3　IAP USART 硬件环境

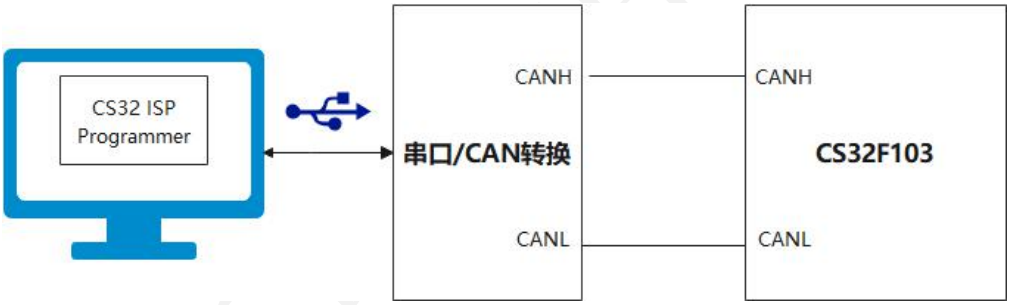

图 4　IAP CAN 硬件环境

2. csboot IAP 应用指导

2.1. 分层结构

|__ Main

main.c :	主函数入口
config.h :	用户配置文件

|__ Startup

startup_cs32f103xb.s :	启动文件

|__ System

system_cs32f10x.c :	系统初始化文件

|__ Application 协议层

csboot.c :	csboot_handle 初始化文件
cmd_process.c	协议层命令处理文件

|__Interface 接口层

flash_interface.c	FLASH 操作接口文件
ob_interface.c	字节选项区操作接口文件
ram_interface.c	RAM 操作接口文件
usart_interface.c	USART 接口处理文件
can_interface.c	CAN 接口处理文件

|__Middleware 驱动应用中间层

common.c	公共变量定义
hal_comm_usart.c	USART 应用接口
hal_comm_can.c	CAN 应用接口

|__Driver 驱动层

2.2. 硬件参考设计

csboot IAP 支持 USART 和 CAN 两种通信接口，引脚选用如下表所示：

表 1 通信接口引脚

通信接口	接收引脚	发送引脚
CAN	PA11 - CAN RX	PA12 - CAN TX
USART	PA10 - USART RX	PA9 - USART TX

USART1 默认启用 115200 波特率；

CAN 默认启用 500K 波特率。

2.3. 芯片空间分配

csboot IAP 程序中默认分配 FLASH 如下：

- 0x08000000 - 0x08009FFF : IAP 程序空间
- 0x0800A000 - 0x0801FFFF: APP 程序空间

2.4. csboot 工作流程

csboot 工作流程大致如下：

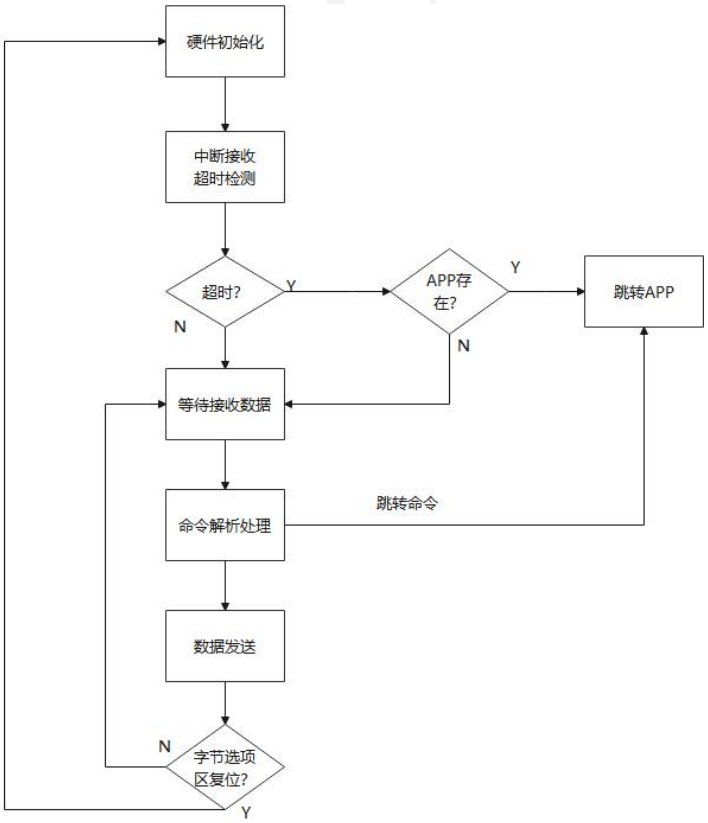

图 5 代码流程

2.5. config.h 配置介绍

FLASH_APP_OFFSET : 应用代码偏移位置，此值必须是 2048 的整数倍

CSBOOT_TIMEOUT_CHECK : 超时检测的时间设置

CSBOOT_COMMUNICATION_INTERFACE_USART
CSBOOT_COMMUNICATION_INTERFACE_CAN : 接口选择

CAN_RX_FRAME_ID CAN 接收 ID（默认接收标准帧）
CAN_TX_FRAME_ID CAN 发送 ID（默认发送标准帧）

Demo 中配置如下图所示：

默认主应用程序在 0X0800A000(40K)位置开始，超时检测默认 1s，以 CAN 为通信接口，接收 ID 为 0x110，发送 ID 为 0X111。

```
/* Main app memory offset. This value must be a multiple of 4096. */
#define FLASH_APP_OFFSET                (uint32_t)0xA000
/* Main app memory start address */
#define FLASH_APP_ADDR                  (FLASH_BASE + FLASH_APP_OFFSET)


/* CSBOOT time-out check(ms) */
#define CSBOOT_TIMEOUT_CHECK            (uint32_t)1000


/* CSBOOT communication interface definitions. Only one interface can be set to 1 */
#define CSBOOT_COMMUNICATION_INTERFACE_USART    0
#define CSBOOT_COMMUNICATION_INTERFACE_CAN      1



/* CAN rx frame ID(default standard ID) */
#define CAN_RX_FRAME_ID                         0x110
/* CAN send frame ID(default standard ID) */
#define CAN_TX_FRAME_ID                         0x111
```

图 6 IAP 配置选择

3. CAN 接口注意事项

(1)修改字节选项区后，需要复位才能生效。CAN 在发送字节选项区的命令后(如读写保护去使能)，下一帧数据应至少间隔 50ms，避免丢帧。

(2)CAN 默认以 500Kbps 速率通信，用户应按应用场合修改。

(3)checksum 在 CAN 通信中作为帧结尾判断。

4. 主应用代码编写指导

4.1. 向量表偏移

主应用代码是指放在 iap 之后的，待跳转的目标代码段。CORTEX-M3 核支持中断向量表的偏移，如下图所示操作即将中断向量表偏移 0XA000。

```
int main(void)
{
    SCB->VTOR = FLASH_BASE | 0xA000;
    /* Configure USART3 as the printing port. */
    usart_config();
    /* Configure NVIC. */
    nvic_config();
    /* Configure TIM1 and TIM2. */
    tim_config();
    while(1);
}
```

图 7　向量表偏移

4.2. 主程序地址偏移

本应用笔记中 0X08000000 - 0X0800A000 之间的 FLASH 存放 IAP 代码，主应用代码只能放在 0X0800A000 之后的 FLASH 空间中，所以用户在编写主 FLASH 代码是应明确 APP 代码的起始地址，避免与 IAP 升级代码地址区间有交叉。

KEIL 用户修改地址参考如下图：

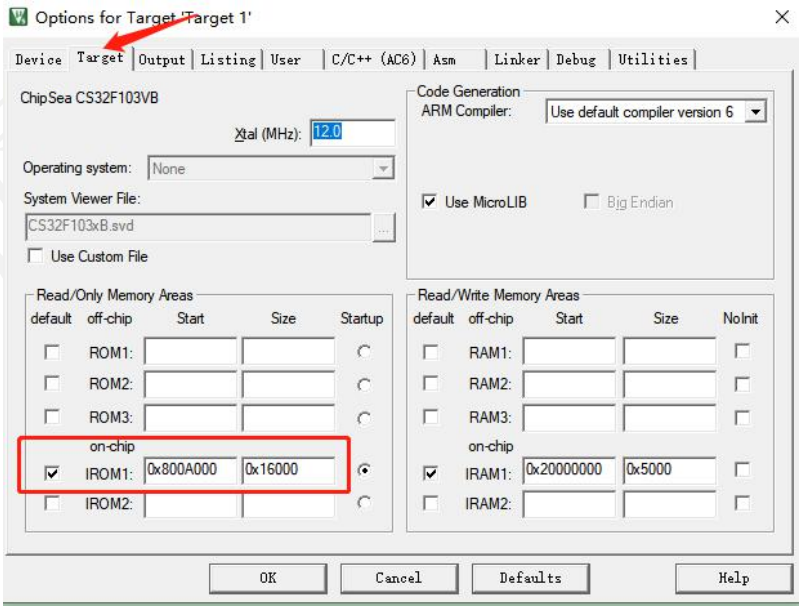

图 8　MDK 主程序偏移

5. csboot 通信协议

IAP 升级方案采用芯海科技私有协议进行信息传输，以下对协议内容进行详细说明。

5.1. 协议帧格式

协议帧根据传输方向可分为两类，第一类为上位机下发至芯片端的指令帧，其帧格式如下表所示：

Head	Length	Command	Data	CKL	CKH
0x55	命令码+数据	命令码	数据	帧校验码低	帧校验码高

第二类协议帧为芯片端上传至上位机的响应帧，其帧格式如下表所示：

ACK	Head	Length	Execute	Data	CKL	CKH
响应状态	0x55	执行状态+	执行状态	数据	帧校验码	帧校验码

协议帧数据长度最大为 255Bytes，最小可为 1Byte，即协议帧可不包含 Data 段，采用累加和不计溢出的方式校验指令帧。

本方案中 ROM 数据传输采用小端模式，即先传输地址低位，再传输地址高位，协议帧数据长度最大为 133Bytes，为 Address(4Bytes) + ROMData(128Bytes)，一帧数据需在 50ms 内下发完成(CAN 协议可加长此值)。

5.2. 协议指令

5.2.1. 命令码

ISP 支持的命令码列表如下表所示：

命令	命令码	描述	响应	备注
GetID	0x20	获取芯片 ID	Yes	
GetVersion	0x21	获取 Bootloader 版本号信息	Yes	
Write Memory	0x23	向 Flash/SRAM 区域写入数据	Yes	
Read Memory	0x24	读取 Flash/SRAM 区域数据	Yes	
ErasePage	0x25	擦除 Flash 某一页	Yes	
Erase Chip	0x26	擦除整个 Main Flash 区域	Yes	
Go	0x27	强制跳转至指定地址	No	
Write Protect	0x28	设置 Flash 扇区写保护	Yes	
Write Unprotect	0x29	解除 Main Flash 所有扇区写保护	Yes	
Readout Protect	0x2A	设置芯片读保护	Yes	
Readout	0x2B	解除芯片读保护	Yes	

5.2.2. 响应状态

响应状态是指目标芯片端对接收到的数据的解析结果，以向上位机反馈协议帧是否下发正确，并对错误类型进行分类。

响应状态	描述	备注
0x00	数据包接收正确（ACK)	
0x11	包头错误	
0x12	包校验错误	
0x13	包长度错误，为 0	
0x14	包长度错误，超过最大允许长度	
0x15	未知错误	

5.2.3. 执行状态

执行状态用于指示目标芯片端对接收到的指令的执行情况，上位机可据此决定后续下发的指令。

命令状态	描述	备注
0x00	命令执行成功	
0x01	无效或不支持的命令	
0x02	命令参数错误	如擦写地址超限、读取数据超界等
0x03	命令无权限	如读写保护等
0x04	命令执行失败	如协议帧校验失败等

5.3. 通信数据

5.3.1. Get ID 命令

Byte0	Byte1	Byte2	Byte3	Byte4
Head	Length	Command	CKL	CKH
0x55	0x01	0x20	0x76	0x00

芯片端回复数据：

Byte0	Byte1	Byte2	Byte3	Byte4-7	Byte8	Byte9
ACK	Head	Length	Execute	ID[0:3]	CKL	CKH
0x00	0x55	0x05	0x00			

注：

ID：芯片 ID，从低位至高位依次发送。

5.3.2. Get Version 命令

命令描述：

获取 Bootloader 版本号（32bits）。

通信示例：

上位机下发数据：

Byte0	Byte1	Byte2	Byte3	Byte4
Head	Length	Command	CKL	CKH
0x55	0x01	0x21	0x77	0x00

芯片端回复数据：

Byte0	Byte1	Byte2	Byte3	Byte4-7	Byte8	Byte9
ACK	Head	Length	Execute	Version[0:3]	CKL	CKH
0x00	0x55	0x05	0x00			

注：

Version：BootLoader 版本号，从低位至高位依次发送。

5.3.3. Write Memory 命令

命令描述：

向指定地址写入指定数据，执行对 Main Flash、SRAM 或 Option Byte 的编程。

通信示例：

上位机下发数据：

Byte0	Byte1	Byte2	Byte3-6	Byte7-	Byte(7+N)	Byte(8+N)
Head	Length	Command	Addr[0:3]	ROM	CKL	CKH
0x55	N+5	0x23				

注：

Addr：数据起始地址，从低位至高位依次发送。

ROM Data：ROM 区数据。

芯片端回复数据：

Byte0	Byte1	Byte2	Byte3	Byte4	Byte5
ACK	Head	Length	Execute	CKL	CKH
0x00	0x55	0x01	0x00	0x56	0x00

5.3.4. Read Memory 命令

命令描述：

读取 Main Flash、SRAM 或 Option Byte 中指定地址处数据。

通信示例：

上位机下发数据：

Byte0	Byte1	Byte2	Byte3-6	Byte7	Byte8	Byte9
Head	Length	Command	Addr[0:3]	Data_Len	CKL	CKH
0x55	0x06	0x24		N		

注：

Addr：数据起始地址，从低位至高位依次发送。

Data_Len：读取数据长度，要求 N<255。

芯片端回复数据：

Byte0	Byte1	Byte2	Byte3	Byte4-	Byte(5+N)	Byte(6+N)
ACK	Head	Length	Execute	ROM	CKL	CKH
0x00	0x55	N+1	0x00			

注：

ROM Data：ROM 区数据。

5.3.5. Erase Page 命令

命令描述：

擦除 Main Flash 某一页数据，此命令无法操作 SRAM 区域。

通信示例：

上位机下发数据：

Byte0	Byte1	Byte2	Byte3-6	Byte7	Byte8
Head	Length	Command	Addr[0:3]	CKL	CKH
0x55	0x05	0x25			

注：

Addr：待擦除页地址，从低位至高位依次发送。

芯片端回复数据：

Byte0	Byte1	Byte2	Byte3	Byte4	Byte5
ACK	Head	Length	Execute	CKL	CKH
0x00	0x55	0x01	0x00	0x56	0x00

5.3.6. Erase Chip 命令

命令描述：

擦除 Main Flash 区域所有数据，不包括 Option bytes 区域。

通信示例：

上位机下发数据：

Byte0	Byte1	Byte2	Byte3	Byte4
Head	Length	Command	CKL	CKH
0x55	0x01	0x26	0x7C	0x00

芯片端回复数据：

Byte0	Byte1	Byte2	Byte3	Byte4	Byte5
ACK	Head	Length	Execute	CKL	CKH
0x00	0x55	0x01	0x00	0x56	0x00

5.3.7. Go 命令

命令描述：

强制跳转至 Main Flash 或 SRAM 中指定地址执行代码，若命令帧校验正确，则芯片端会立即执行跳转且不做响应回复。

通信示例：

上位机下发数据：

Byte0	Byte1	Byte2	Byte3-6	Byte7	Byte8
Head	Length	Command	Addr[0:3]	CKL	CKH
0x55	0x05	0x27			

注：

Addr：跳转地址。

芯片端回复数据：无。

5.3.8. Write Protect 命令

命令描述：

设置 Main Flash 扇区写保护功能，修改完成后将会触发系统复位，使新设置生效。

通信示例：

上位机下发数据：

Byte0	Byte1	Byte2	Byte3-10	Byte11	Byte12
Head	Length	Command	WRP	CKL	CKH
0x55	0x09	0x28			

注：

WRP：写保护数据，其数据组成结构如下：

WRP0	nWRP0	WRP1	nWRP1	WRP2	nWRP2	WRP3	nWRP3

其中，nWRP 为 WRP 反码，WRP 数据与保护页对应关系示例如下：

WRP	Value	Protect address	Page
WRP0	0b11111110	0x08000000 - 0x08000FFF	Page0 - page3
	0b11111101	0x08001000 - 0x08001FFF	Page4 - page7
	011111011	0x08002000 - 0x08002FFF	Page8 - page11
	0b11110111	0x08003000 - 0x08003FFF	Page12 - page15
	0b11101111	0x08004000 - 0x08004FFF	Page16 - page19
	0b11011111	0x08005000 - 0x08005FFF	Page20 - page23
	0b10111111	0x08006000 - 0x08006FFF	Page24 - page27
	0b01111111	0x08007000 - 0x08007FFF	Page28 - page31
WRP1	0b00000000	0x08008000 - 0x0800FFFF	Page32 - page63
WRP2	0b00000000	0x08010000 - 0x08017FFF	Page64 - page95
WRP3	0b00000000	0x08018000 - 0x0801FFFF	Page96 - page127

芯片端回复数据：

Byte0	Byte1	Byte2	Byte3	Byte4	Byte5
ACK	Head	Length	Execute	CKL	CKH
0x00	0x55	0x01	0x00	0x56	0x00

5.3.9. Write Unprotect 命令

命令描述：

解除所有 Main Flash 扇区的写保护功能，修改完成后将会触发系统复位，使新设置生效。

通信示例：

上位机下发数据：

Byte0	Byte1	Byte2	Byte3	Byte4
Head	Length	Command	CKL	CKH
0x55	0x01	0x29	0x7F	0x00

芯片端回复数据：

Byte0	Byte1	Byte2	Byte3	Byte4	Byte5
ACK	Head	Length	Execute	CKL	CKH
0x00	0x55	0x01	0x00	0x56	0x00

5.3.10. Readout Protect 命令

命令描述：

设置 Main Flash 读保护功能，修改完成后将会触发系统复位，使新设置生效。

通信示例：

上位机下发数据：

Byte0	Byte1	Byte2	Byte3	Byte4
Head	Length	Command	CKL	CKH
0x55	0x01	0x2A	0x80	0x00

芯片端回复数据：

Byte0	Byte1	Byte2	Byte3	Byte4	Byte5
ACK	Head	Length	Execute	CKL	CKH
0x00	0x55	0x01	0x00	0x56	0x00

5.3.11. Readout Unprotect 命令

命令描述：

解除 Main Flash 读保护功能，修改完成后将会触发系统复位，使新设置生效。

通信示例：

上位机下发数据：

Byte0	Byte1	Byte2	Byte3	Byte4
Head	Length	Command	CKL	CKH
0x55	0x01	0x2B	0x81	0x00

芯片端回复数据：

Byte0	Byte1	Byte2	Byte3	Byte4	Byte5
ACK	Head	Length	Execute	CKL	CKH
0x00	0x55	0x01	0x00	0x56	0x00

5.4. 异常回复

在帧接收异常或执行异常后，芯片端可按此格式回复异常情况：

Byte0	Byte1	Byte2	Byte3	Byte4	Byte5
ACK	Head	Length	Execute	CKL	CKH
	0x55	0x01			

上位机在接收到异常回复后应立即停止发送数据，并等待 20ms 后才可重新发送数据。

6. 上位机应用指导

芯海 IAP 的协议指令与 ISP 指令相同，可共用 CS32 ISP Programmer 开发工具。所不同的是，采用 ISP 需要通过跳线 boot 引脚来切换到 boot 启动，IAP 则省去此步骤。

目前芯海提供的编程工具仅支持 USART 通信接口，其它接口暂不支持。

6.1. 上位机地址写入起始地址

不同用户设置的 IAP 空间大小不同，上位机中需要修改主应用代码的起始地址。Config.ini 中找到 CS32F103xB 型号，修改 AppAddr 和 AppSize 两个参数。

```
[CS32F103xB 128K]
PageSize=1024
FlashSize=0x20000
AppAddr=0x0800A000
AppSize=0x00016000
DataAddr=0x00000000
DataSize=0x00000000
OptionAddr=0x1FFFF800
OptionSize=0x00000010
DeviceConfig=115200,8,1,2
OptionConfig=CS32FX103-128K.xml
ProtocolConfig=CS32F103UartProtocol
```

图 9　修改起始地址

6.2. IAP 程序延时参数

在 IAP 程序中的 config.h 文件中，宏 CSBOOT_TIMEOUT_CHECK 定义了 IAP 等待握手指令的时间。由于人工点击下载按钮的时间，此宏定义不宜过短。

6.3. 上位机操作步骤

- 先将 CS32F103 复位。
- 在宏 CSBOOT_TIMEOUT_CHECK 定义的时间内点击“下载程序”
- 等待程序下载完成

图 10 下载程序

免责声明和版权公告

本文档中的信息，包括供参考的 URL 地址，如有变更，恕不另行通知。

本文档可能引用了第三方的信息，所有引用的信息均为“按现状”提供，芯海科技不对信息的准确性、真实性做任何保证。

芯海科技不对本文档的内容做任何保证，包括内容的适销性、是否适用于特定用途，也不提供任何其他芯海科技提案、规格书或样品在他处提到的任何保证。

芯海科技不对本文档是否侵犯第三方权利做任何保证，也不对使用本文档内信息导致的任何侵犯知识产权的行为负责。本文档在此未以禁止反言或其他方式授予任何知识产权许可，不管是明示许可还是暗示许可。

Wi-Fi 联盟成员标志归 Wi-Fi 联盟所有。蓝牙标志是 Bluetooth SIG 的注册商标。

文档中提到的所有商标名称、商标和注册商标均属其各自所有者的财产，特此声明。

芯海科技
CHIPSEA
股票代码:688595