Technical Article

基于 CC2340R5 的 bootloader 固件更新上位机实现

Gao Wei, Albin Zhang, Wesley H, Leon Yan

1. 摘要:

CC2340R5 是 TI 的超低功耗无线 MCU ，为各种应用提供了高性价比的解决方案。芯片由主处理器 M0+、软件可编程射频处理器、电源管理和丰富的外设构成片上系统。可以实现蓝牙，Zigbee， Thread，私有无线等通信网络协议。具有集成度高、宽温度范围、低成本、低功耗、优秀的射频性能等特点。

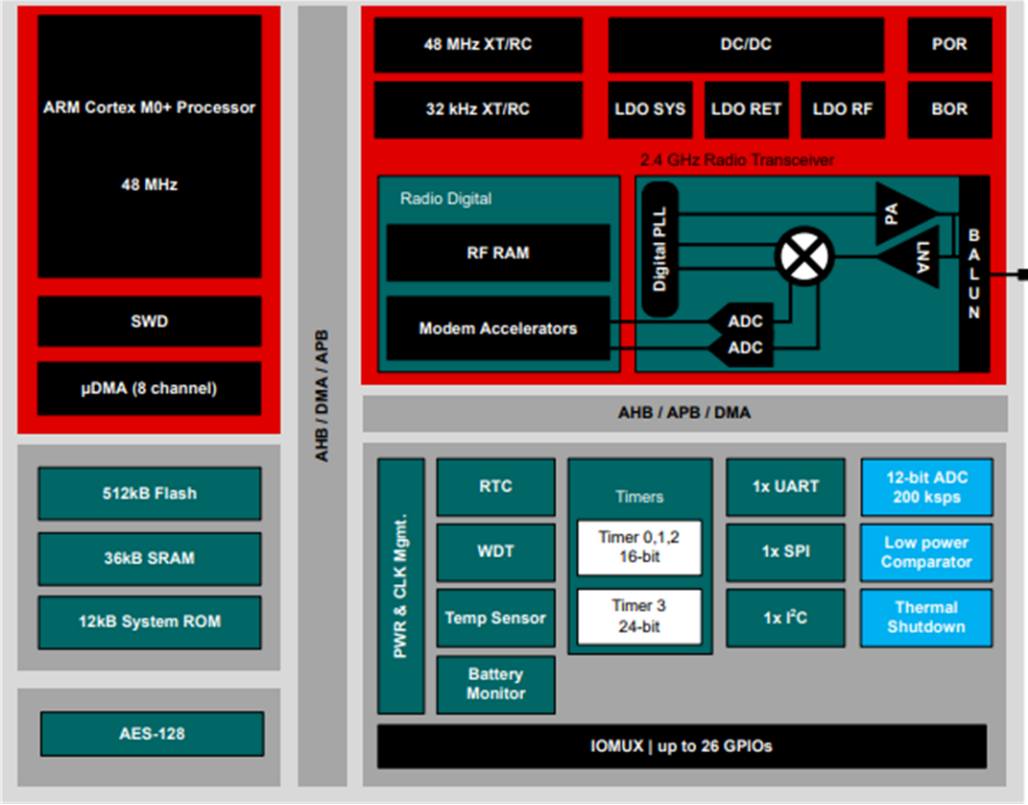

图 1-1 CC2340 芯片框图

在 CC2340 系列产品使用过程中，不少用户希望通过 CC2340 内部固化的 Bootloader 来进行固件程序的更新或者实现产线离线烧录的功能，本用户指南重点介绍可为 CC23x 系列 Bootloader 更新 APP 的上位机设计，本文上位机基于 CC2340R5 讲解如何通过 UART 接口与 Bootloader 通信实现固件更新。附录为基于 MCU 的上位机伪代码实现，可以按本文讲解的设计流程在不同 MCU 平台中实现自主设计。

2. CC2340 ROM Serial Bootloader:

Boot 流程:

CC2340 上电 Boot 过程中有 CCFG 和 FCFG 两个文件与 Bootloader 相关，CCFG 是用户自定义配置文件，FCFG 则是 TI 出厂默认配置文件提供默认的 Bootloader，CCFG 支持配置 Bootloader 和 Trigger pin 功能。如果启用了 Bootloader 并且将 Trigger pin 的相关引脚设置为正确的逻辑电平，则 Bootloader 会被启动。激活引导加载程序后，会在上电复位 10ms 后与外部主机通信。 ROM Bootloader 支持通过可自定义引脚的 SPI 或 UART 对设备进行固件更新，因此用户如果无此更新需求，出于安全原因，也可以完全禁用 引导加载程序。下图 2-1 所示为 CC2340 启动代码的简化流程图。

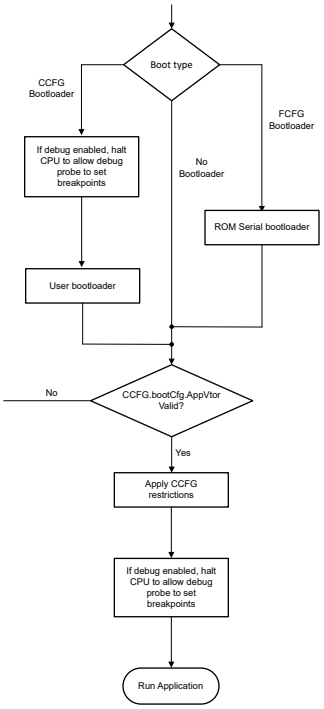

图 2-1 CC2340 Boot Flow

SysConfig 配置 CCFG:

在 SysConfig 中, 可以选择 CCFG 来配置 Bootloader、通信 IO 接口引脚、trigger pin 的使能以及电平设置。

根据封装不同，Bootloader 的管脚是固定的，客户在制备原理图及治具的时候需要特别注意。

Signal	serialIoCfgIndex == 0	serialIoCfgIndex == 1	serialIoCfgIndex == 2
UART_RX	DIO20	DIO12	DIO22
UART_TX	DIO6	DIO13	DIO20
SPI_CLK	DIO8	DIO24	DIO24
SPI_CS	DIO11	DIO11	DIO11
SPI_POCI	DIO12	DIO21	DIO12
SPI_PICO	DIO13	DIO13	DIO13

图 2-2 CC2340 UART/SPI

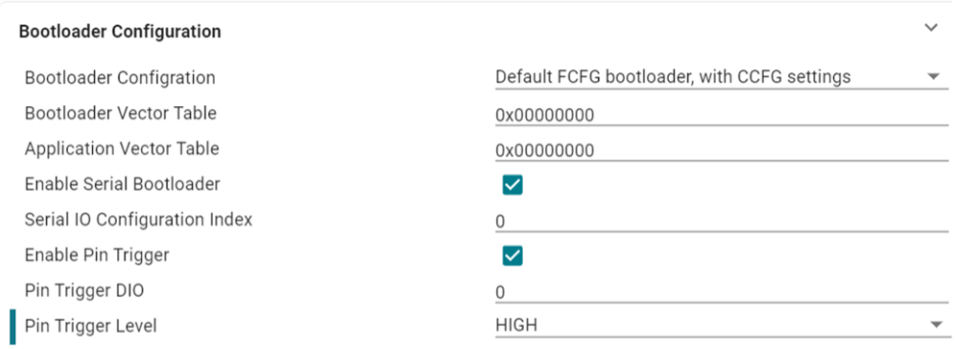

图 2-2 CC2340 Bootloader Configuration

上述 SysConfig 配置即在 DIO0 引脚电平被拉高时，Bootloader 会被激活，会在上电复位时与外部主机通信，通信接口 index 为 0 对应 pin 脚如图 2-2 中第一列。用户可以根据实际使用过程中资源情况来配置 Bootloader 相关参数，为后续与上位机通信做准备。

3. 上位机与 Bootloader 通信规则:

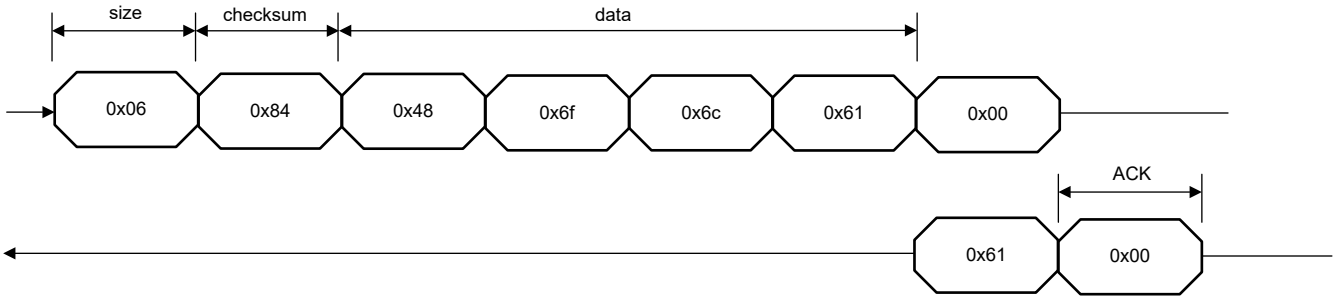

图 3-1 CC2340 Bootloader Data Format

ROM Bootloader 通信规则详见 CC23x 系列 TRM 中 8.5 章节。主机需要通过特定的数据包格式向 Bootloader 传输特定的命令才能完成特定的操作，数据包由数据包长度、数据包校验和、数据段组成；如上图中所示主机发送了一串 0x06, 0x84, 0x48, 0x6f, 0x6c, 0x61 数据包，数据包中第 1 个字节 0x06 为该数据包总长度；第 2 个字节 0x84 为数据包校验和，计算方式是将数据段所有字节相加后取低 8 bits，即 0x48 + 0x6 f+ 0x6c + 0x61 = 0x184 取 0x184 低 8bits 即为 0x84；数据段主要有两种表现形式：bootloader 控制命令码、固件更新数据。

CC2340 Bootloader 上位机控制命令 ID:

CMD ID	Value
BLDR_CMD_PING	0x20
BLDR_CMD_GET_STATUS	0x21
BLDR_CMD_GET_PART_ID	0x22
BLDR_CMD_RESET	0x23
BLDR_CMD_CHIP_ERASE	0x24
BLDR_CMD_CRC32	0x25
BLDR_CMD_DOWNLOAD	0x26
BLDR_CMD_DOWNLOAD_CRC	0x27
BLDR_CMD_SEND_DATA	0x28

图 3-2 CC2340 Bootloader Command IDs

如上表所示，Bootloader 共有 9 种控制命令码，包括复位、擦除、CRC、下载、数据发送等。

通信数据包可以分为三个 Part：

Part 1	Part 2	Part 3
数据包长度	数据校验和	1.CMD_ID
		2.固件数据(下载 APP/CCFG 时)

CC2340 Bootloader 应答状态格式：

上位机的命令发出后，Bootloader 接收并校验后会对上位机做出应答，接收成功即为 ACK, 接收失败即为 NACK。

Protocol Byte	Value
ACK	0xCC
NACK	0x33

图 3-3 CC2340 Bootloader Return Type

Bootloader 固件更新流程：

首先，需要触发从机进入 Bootloader 模式等待上位机发起通信进行固件传输更新，有两种方式能让芯片进入 Bootloader：擦空芯片、触发 Trigger pin 电平。从机进入 Bootloader 后，上位机的控制流程分为：握手、擦除、APP 下载、CCFG 下载、复位，如下图所示：

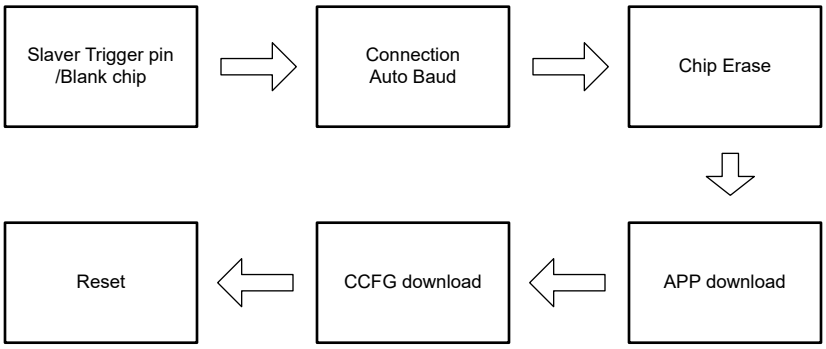

图 3-4 CC2340 Host and Bootloader Communication Flow

4. ROM Serial Bootloader:

Trigger pin 启用：

从机 Trigger pin 使能后，进行固件更新前需要先根据配置的触发电平置高或置低 Trigger pin，随后复位 MCU 即会进入 Bootloader 等待上位机通信，此时可以恢复 Trigger pin 电平状态。

建立 UART 通信：

上位机首先需要与 Bootloader 进行握手，建立通信基础，根据 CC2340 TRM Chapter8 内容可知上位机需连续发送 0x55,0x55 两个字节, 收到 Bootloader 返回的 ACK（0,0xCC 两字节）时即代表双方握手成功，可以进行下一步操作。

注：主机可以根据需要设置合适的波特率，Bootloader 在握手阶段会自动识别主机端的波特率，但波特率不能超过 CC2340 TRM 中规定的 1.6Mbps。

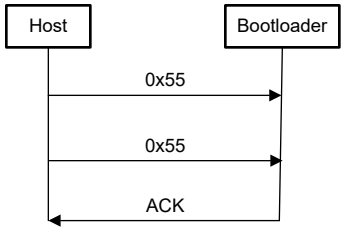

图 4-1 CC2340 Host and Bootloader Connect

主机与 Bootloader 握手数据包：

0x55	0x55

flash 擦除：

擦除 CMD 格式：

Byte	Field	Description
0	cmdId	BLDR_CMD_CHIP_ERASE command ID

握手数据包发送后主机若回读到 Bootloader 的 ACK 信号，便可下发擦除命令进行烧录前的 flash 擦除。主机控制 Bootloader 擦除 flash 命令数据包：

Length	Checksum	CMD_ERASE
0x03	0x24	0x24

APP 数据下载：

下载 CMD 格式：

Byte	Field	Description
0	cmdId	BLDR_CMD_DOWNLOAD command ID
1	Program Address [31:24]	Start address of the download
2	Program Address [23:16]	
3	Program Address [15:8]	
4	Program Address [7:0]	
5	Program Size [31:24]	Number of bytes (length of the download)
6	Program Size [23:16]	
7	Program Size [15:8]	
8	Program Size [7:0]	

图 4-2 CC2340 CMD_DOWNLOAD Format

同样的，当擦除命令下发后回读到 Bootloader 的 ACK 信号，便可下发 APP 下载命令并进行 APP 数据的传输。APP 下载分两步，首先下发下载命令，紧接着循环发送 APP 数据直至发完所有 APP 数据。

下载命令 BLDR_CMD_DOWNLOAD 共由 9 个 Byte 组成，其中包括了 ID、数据下载的起始地址、数据总长度三个部分。

首先上位机下发下载命令 CMD_DOWNLOAD，如下数据包所示，APP 下载起始地址 0x00000000，APP 长度为 0x00003200:

Length	Checksum	CMD_DOWNLOAD								
0x0B	0x58	0x26	0x00	0x00	0x00	0x00	0x00	0x00	0x32	0x00

当上位机下载命令下发后成功回读到 ACK 就可以开始进行 APP 数据的传输，由于数据包长度是通过一个字节空间来控制，所以每个数据包最多能传输 252 Byte 实际数据（255Byte – 1Byte Length – 1Byte Checksum – 1Byte CMD_ID = 252Byte），故当 APP 数据大于 252Byte 时就需要循环传输数据包，直至如上规定的总长度为 0x3200 的所有 APP 数据被全部传输完成。

APP 下载流程如下图所示：

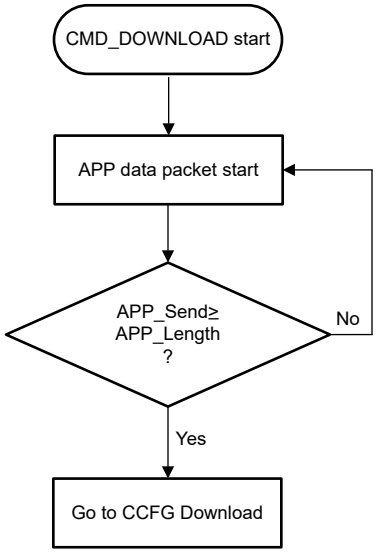

图 4-3 CC2340 APP Download Flow

CCFG 数据下载：

当 APP 数据全部传输完成后，按照同样的步骤进行 CCFG 数据的传输。

首先，上位机下发下载命令 CMD_DOWNLOAD:

Length	Checksum	CMD_DOWNLOAD								
0x0B	0x7E	0x26	0x4E	0x02	0x00	0x00	0x00	0x00	0x08	0x00

如上数据包所示，CCFG 文件下载至起始地址 0x4E020000，文件长度为 0x800（2k Byte），对于 CCFG 而言，下载起始地址和长度都是固定的。

当上位机下载命令下发后成功回读到 ACK 就可以开始进行 CCFG 数据的循环传输，具体操作流程与 3.3 中 APP 下载流程一致。

RESET:

CCFG 文件传输完成后，所有数据传输就结束了，上位机需要下发复位命令控制从机复位即可运行更新后的代码文件。复位命令数据包如下：

Length	Checksum	CMD_ERASE
0x03	0x23	0x23

5. BLDR_GET_STATUS 命令使用注意事项:

上位机在下发操作命令后若 Bootloader 成功应答 ACK（此 ACK 代表接收成功），上位机可以发送 GET_STATUS 命令查询从机上一条命令是否执行完成，若从机应答 ACK + RET_STATUS 且 RET_STATUS（应答状态码）为 0x40 即代表上一条命令已成功执行。需要特别注意的是，主机在接收到正确的 ACK + RET_STATUS 后需要向 Bootloader 发送 ACK 应答。一次成功的使用 GET_STATUS 案例如下图所示：

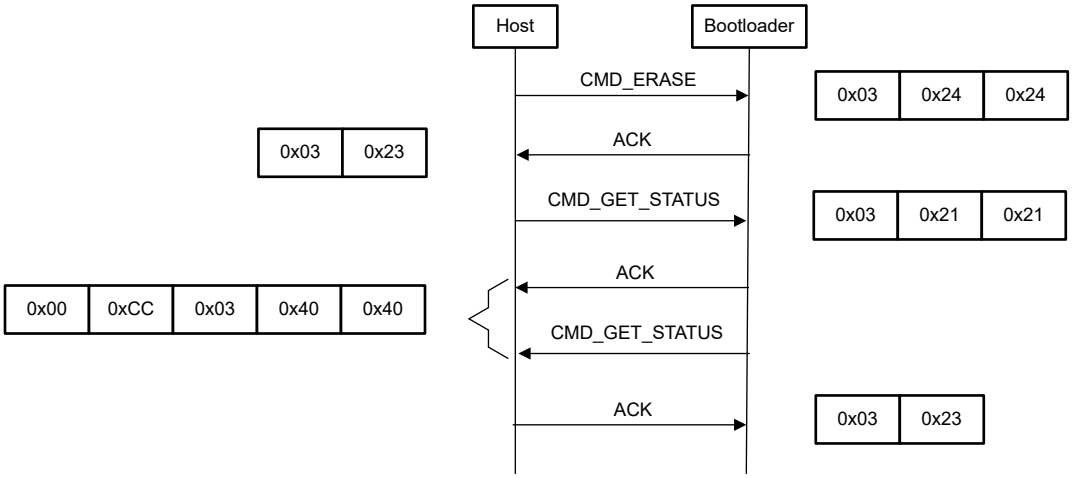

图 5-1 CC2340 CMD_ERASE + GET_STATUS Example

如下图所示是逻辑分析仪抓取一次完整 CMD 通信的 UART 总线数据包：

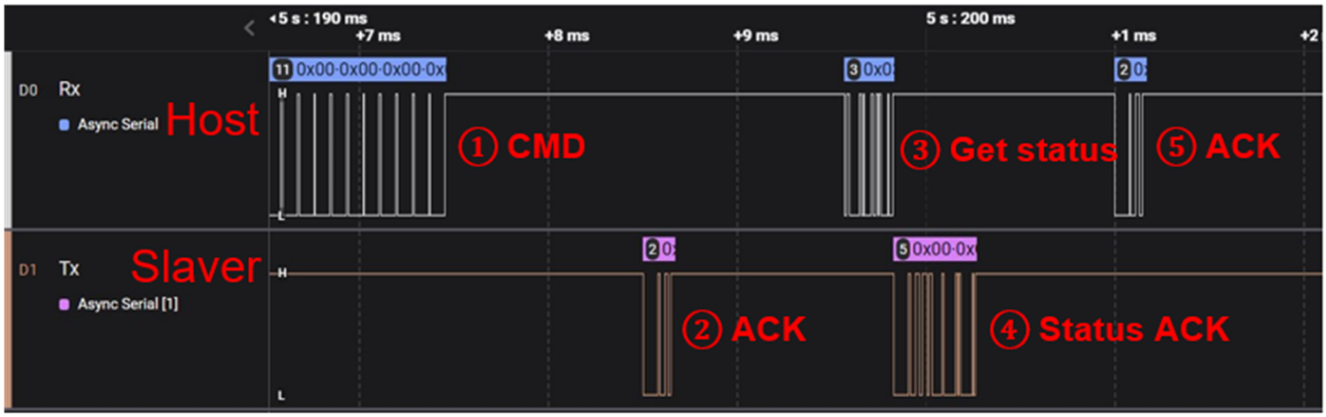

图 5-2 UART Bus Data Communication Example

6. 总结:

本文讲解了 CC2340 系列无线 MCU 的 ROM Serial Bootloader 固件更新方法，分步解析了 Bootloader 上位机通信的各个环节，为用户自主设计上位机提供了基本框架，CC2340 固化于 ROM 中的 Bootloader 可以为用户提供较为灵活的硬件接口、通信方式自定义配置，用户通过本文的讲解可以快速搭建基于 MCU 的上位机。此外，快速上手 Bootloader 的通信规则后用户可以结合自己的实际情况在不同的平台下搭建上位机，实现 CC2340 系列无线 MCU 便捷、低成本的固件更新方案。

7. 参考文献:

1. CC23xx SimpleLink Wireless MCU Technical Reference Manual
2. CC2538/CC26x0/CC26x2 Serial Bootloader Interface (Rev. D)
3. CC13x0/CC26x0 SimpleLink Wireless MCU Technical Reference Manual
4. Initial release of cc23xx examples in ble_examples github

8 附录

8.1 部分伪代码

a.主状态机控制

```
voidFunction_BootSeq(UART2_Handle uart)
{
     uint32_t length,lengthc;
     staticchar *i;
     static uint32_t j,g_flag_APPDone;
     staticchar *p = (char*)g_AddrStarAPP;
     staticchar *pc = (char*)g_AddrStarCCFG;
     length = g_LengthAPP;
     lengthc = g_LengthCCFG;
     switch(sequence)
     {
         caseStatus_BootConect :
             Function_Singlecmd(uart,CMD_AUTO_BAURD,2);
             break;
         caseStatus_BootErase:
             Function_Singlecmd(uart,CMD_CHIP_ERASE,3);
             break;
         caseStatus_BootAPPReq:
             Function_Singlecmd(uart,CMD_APP_DOWNLOADReq,0x0B);
             break;
         caseStatus_BootAPPSend:
             g_flag_APPDone = 0;
             while(!g_flag_APPDone){
                 Function_SendPackage(uart, Uart_SendBatch, p, length);
                 g_flag_APPDone = 1;
                 sequence += 1;
             }
             break;
         caseStatus_BootCCFGReq:
             Function_Singlecmd(uart,CMD_CCFG_DOWNLOADReq,0x0B);
             break;
         caseStatus_BootCCFGSend:
             g_flag_APPDone = 0;
             while(!g_flag_APPDone){
                 Function_SendPackage(uart, Uart_SendBatch, pc, lengthc);
                 g_flag_APPDone = 1;
                 sequence += 1;
             }
             break;
         caseStatus_BootDone:
             Function_Singlecmd(uart,CMD_RESET,3);
             g_flag_button = 0;
             sequence = 0;
             break;
         default:
             break;
         }
     }
```

b.命令下发函数

```
voidFunction_Singlecmd(UART2_Handle uart, const uint8_t *Singlecmd, size_t size_tt)
{
     size_t bytesRead;
     size_t bytesWritten = 0;
     char input[2];

     bytesRead = 0;

         UART2_write(uart, Singlecmd, size_tt, &bytesWritten);
         UART_Read(uart,&input,sizeof(input),&bytesRead);
         if(input[0] == CMD_ACKBYTE1 && input[1] == CMD_ACKSUCCESS)
         {
     sequence += 1;
 }
}
```

c.固件数据下载函数

```
void Function_SendPackage(UART2_Handle uart, uint8_t *buffer, char *pp, uint32_t length)
{
    static char *i;
    static uint32_t j;
    for(i = pp;i < (pp + length);)
    {
        for(j = 0;j< DATAPACK_SIZE;j++)
        {
            if(i < (pp + length))
            {
                buffer[j] = *i;
                i++;
            }
        }
        Function_SendData(uart, (const uint8_t *)buffer);
    }
}
```

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265