# 基于 UDS 的 Bootload 使用手册

V0.1

文档修订历史：

| 日期 | 作者 | 更新内容 | 备注 |
| --- | --- | --- | --- |
| 2023-03-30 | TOSUN | 创建文档 | |
| | | | |

上海同星智能科技有限公司

2023 年 03 月

目录

# 简介

本文介绍英飞凌 TLE989X 系列的一种通用的 Bootloader 实现方法，Bootloader 通过任意通信口都可以远程升级产品固件（程序），解决了 MCU 烧写程序需要拆解设备或者需要专业人员、专用工具、以及现场操作的烦恼。本次提供的 Bootloader 集成了部分 UDS（14229、15765 规范）服务结合 TSMaster 上位机通过 CANFD 接口实现下载 APP 程序。

# 概述

本文介绍的内容包括：如何通过 Bootload 配合上位机下载 APP 程序。TSMaster 作为 Bootloader 的上位机，使用 UDS 协议传输 APP.HEX 文件到单片机(底层通讯协议为 CANFD);Bootloader 程序解析上位机传输过来的数据包，并将 APP 代码包组合起来，按照顺序写入到目标 Flash 空间内；Bootloader 程序引导目标 Flash 区的 APP 程序启动成功后，自动退出运行；APP 程序开始工作。该过程流程图如下图 1：

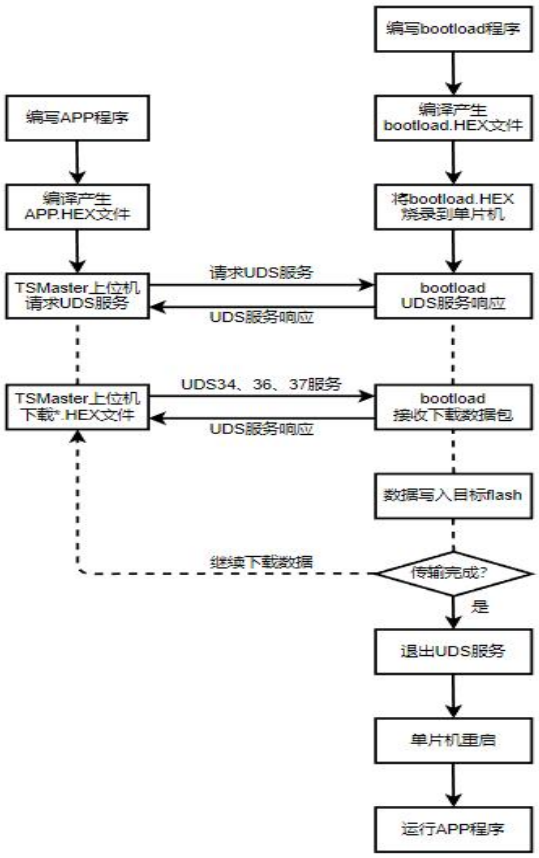


图 1

# 1. UDS

## 1.1UDS 简介

UDS（Unified Diagnostic Services，统一的诊断服务）诊断协议是在汽车电子 ECU 环境下的一种诊断通信协议，在 ISO14229 中规定。目前市面上的新车都具有用于车外诊断的诊断接口，这使得我们可以用诊断工具连接到车辆的总线系统上。因此，UDS 中定义的消息可以发送到支持 UDS 服务的控制器（业内称 ECU）。这样我们就可以访问各个控制单元的故障存储器或用新的固件更新 ECU 的程序。

## 1.2UDS 常用服务

| 大类 | SID（0x） | 诊断服务名 | 服务Service |
|---|---|---|---|
| 诊断和通信管理功能单元 | 10 | 诊断会话控制 | Diagnostic Session Control |
| | 11 | ECU复位 | ECU Reset |
| | 27 | 安全访问 | Security Access |
| | 28 | 通讯控制 | Communication Control |
| | 3E | 待机握手 | Tester Present |
| | 83 | 访问时间参数 | Access Timing Parameter |
| | 84 | 安全数据传输 | Secured Data Transmission |
| | 85 | 控制DTC的设置 | Control DTC Setting |
| | 86 | 事件响应 | Response On Event |
| | 87 | 链路控制 | Link Control |
| 数据传输功能单元 | 22 | 通过ID读数据 | Read Data By Identifier |
| | 23 | 通过地址读取内存 | Read Memory By Address |
| | 24 | 通过ID读比例数据 | Read Scaling Data By Identifier |
| | 2A | 通过周期ID读取数据 | Read Data By Periodic Identifier |
| | 2C | 动态定义标识符 | Dynamically Define Data Identifier |
| | 2E | 通过ID写数据 | Write Data By Identifier |
| | 3D | 通过地址写内存 | Write Memory By Address |
| 存储数据传输功能单元 | 14 | 清除诊断信息 | Clear Diagnostic Information |
| | 19 | 读取故障码信息 | Read DTC Information |
| 输入输出控制功能单元 | 2F | 通过ID控制输入输出 | Input Output Control By Identifier |
| 例行程序功能单元 | 31 | 例行程序控制 | Routine Control |
| 上传下载功能单元 | 34 | 请求下载 | Request Download |
| | 35 | 请求上传 | Request Upload |
| | 36 | 数据传输 | Transfer Data |
| | 37 | 请求退出传输 | Request Transfer Exit |
| | 38 | 请求文件传输 | Request File Transfer |



图 2

## 1.3UDS 下载程序流程

```
Step1：10 03                    //10 服务切换到 03 扩展模式
Step2: 85 02                    //关 DTC(空服务，没有具体实现)
Step3: 28 03 01                 //服务关报文(空服务，没有具体实现)
Step4: 10 02                    //10 服务切换到 02 编程会话
Step5: 27 01                    // 27 服务，解锁，通过安全验证。
Step6: 27 02
Step7: 2e 00 00
Step8: 31 00 00
Step9:(34、36、37）服务        //下载 APP
Step10:11                       //ECU 复位
```

# 2.英飞凌 TLE989X 系列单片机板卡简介

## 2.1 英飞凌 TLE989X 系列单片机储存空间映射

英飞凌 TLE989X 系列单片机的存储空间的设计是按照线性地址排列的，这样做的好处是 RAM、ROM、Flash、寄存器的寻址更为方便和直观。IROM1 的物理地址范围是从 0x11000000——0x11006000,IROM2 的物理地址范围是 0x12002000——0x12040000,IRAM1 的物理地址范围是 0x18000000——0x18002000，IRAM2 的物理地址范围是 0x18002000——0x18007C00。具体情况如下图 3 所示。

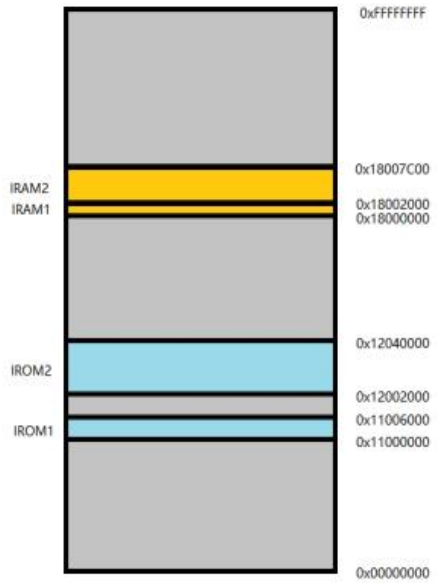


图 3

## 2.2 英飞凌 TLE989X 系列单片机的代码存储分布

英飞凌 TLE989X 系列单片机的 Bootloader 放在从 0x11000000 地址开始的 Flash 的低地址空间内，MCU 上电后会自动从 Bootloader 启动去检查 APP 程序代码是否存在，存在就跳转到 APP 去执行，如果不存在就进入 bootload 等待 TSMster 发起 UDS 服务请求。APP 程序放在 Flash 后面的一个区域内，本例程 APP 程序存放从 0x12002000 地址开始。实际上，这个起始地址也可以改成其他以 1KByte 为单位的起始地址，只要不和 Bootloader 区重合，剩余的 Flash 空间也足够存放 APP 程序即可。Bootload 和 APP 的 ld 文件如下图 4 和下图 5 所示：

```
LR_IROM1 0x11000000 0x00006000  {    ; load region size_region
  ER_IROM1 0x11000000 0x00006000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
   .ANY (+XO)
  }
  RW_IRAM1 0x01800008 0x00001FF8  {  ; RW data
   .ANY (+RW +ZI)
  }
  RW_IRAM2 0x18002000 0x00005C00  {
   .ANY (+RW +ZI)
  }
}

LR_IROM2 0x12002000 0x0000D000  {
  ER_IROM2 0x12002000 0x0000D000  {  ; load address = execution address
   .ANY (+RO)
  }
}
```

图 4

```
LR_IROM2 0x12010000 0x0002E000  {    ; load region size_region
  ER_IROM2 0x12010000 0x0002E000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
   .ANY (+XO)
  }
  RW_IRAM1 0x18000000 0x00002000  {  ; RW data
   .ANY (+RW +ZI)
  }
  RW_IRAM2 0x18002000 0x00005C00  {
   .ANY (+RW +ZI)
  }
}
```

图 5

## 2.3、英飞凌 TLE989X 系列单片机中断向量表

中断向量表在程序启动时和执行中断服务函数的过程中是必不可少的，它相当于程序的“目录”。尤其特别的是 0x00000100——0x00000103 保存的是堆栈空间 MSP——栈顶指针的值。另外，0x0000 0104——0x0000 0107 保存的是 Reset_Handler 函数的指针。截取部分部分 Vector.c 文件内容，如下图 6 所示：

```
__Vectors       DCD     __initial_sp
                DCD     Reset_Handler
                DCD     NMI_Handler               ; NMI Handler
                DCD     HardFault_Handler         ; Hard Fault Handler
                DCD     MemManage_Handler         ; MPU Fault Handler
                DCD     BusFault_Handler          ; Bus Fault Handler
                DCD     UsageFault_Handler        ; Usage Fault Handler
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     0                         ; Reserved
                DCD     SVC_Handler               ; SVCall Handler
                DCD     DebugMon_Handler          ; Debug Monitor Handler
                DCD     0                         ; Reserved
                DCD     PendSV_Handler            ; PendSV Handler
                DCD     SysTick_Handler           ; SysTick Handler
```

图 6

当 Bootloader 引导 APP 程序启动的时候，需要使用 APP 程序的中断向量表。APP 启动后需要将 Bootloader 的向量表切换为 APP 程序自己的向量表，这个过程称为向量表重映射。常规程序中，中断产生后，在做好中断现场压栈工作后，硬件自动寻址对应的中断服务函数入口，并跳转到中断函数执行。跳转代码如下图 7：

```
PMU_serviceFailSafeWatchdogSOW();
GPIO->P1_OMR.reg = 0x00010001;

/* Disable all interrupts */
__disable_irq();

/* point VTOR to new vector table */
CPU->VTOR.reg = USER_APPLICATION_VTAB_ADDRESS;

/*Jump to new application */
BootJumpASM( Address[ 0 ], Address[ 1 ] ) ;
```

图 7

## 2.4 下载 APP 程序流程

单片机上电启动，初始化 CAN,延时 50 毫秒等待是否进入 Bootload 模式。如果不进入 Bootload 模式，检测 0x1200 2000 地址处是否有 APP 程序，如果 APP 存在且无 UDS 服务请求，单片机重启，跳转 0x1200 2000 地址处执行 APP 程序。如果没有检测到 APP 程序存在且无 UDS 服务请求,单片机会进入 Bootload 模式响应上位机发起的 UDS 服务（例如 24、26、27 下载服务）请求。直到上位机发起 11 服务（单片机复位）请求，单片机复位，去检测 APP 代码是否下载完成，如果 APP 下载完成，就会退出 bootload 模式,跳转 0x1200 2000 处执行 APP 代码程序,如果没下载完成，当前状态会依旧是 bootload 模式。如果 APP 程序之前下载过，无需断电重启，可以在 APP 模式下直接重新下载 APP 程序。如果 APP 存在错误，可以断电重启，在 50ms 延时结束之前，发送进入 bootload 模式指令，重新下载 APP。Bootload 下载 APP 流程图如下图 8：

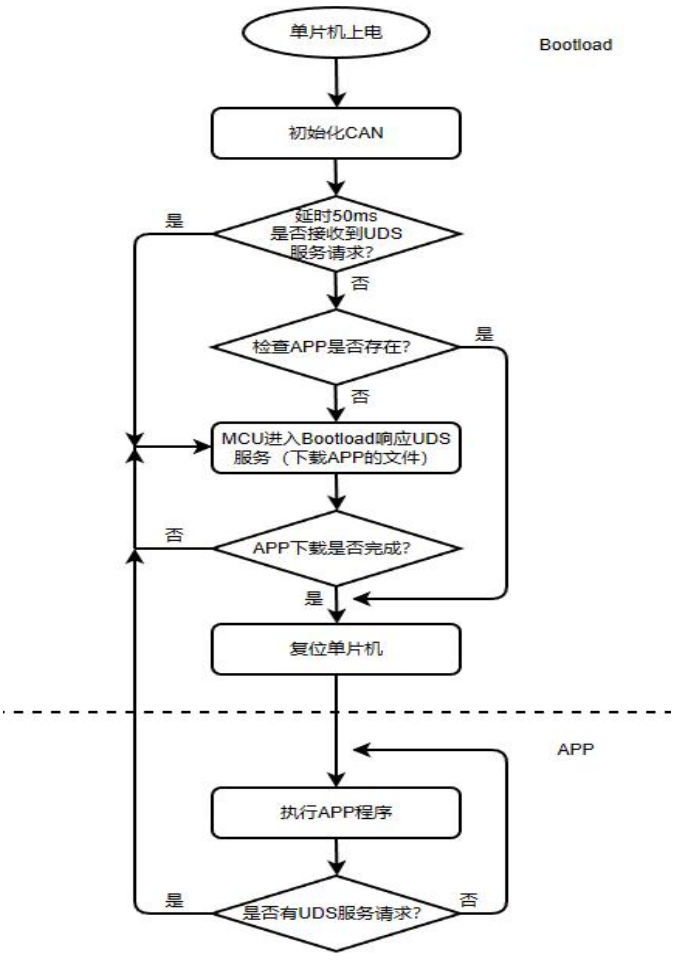


图 8

## 2.5MCU 状态介绍

拿到一块新的的芯片，MCU 处于状态 0，接着开始下载 bootload 代码，当 bootload 代码下载成功就会进入状态 1，bootload 代码开始工作。MUC 进入状态 1 后，就可以开始下载 APP 代码。如果 APP 代码下载失败，就会进入状态 3，此时 MCU 处于 bootload 模式，可以重新尝试下载 APP 程序。如果 APP 程序下载成功，就会进入状态 2，MUC 从 bootload 模式跳转到 APP 模式，APP 开始工作。如果重新下载 APP2 的流程被打断或者发生错误，此时 MCU 会重新进入状态 3，等待重新下载 APP 代码，直至重新下载 APP 成功，MCU 会重新进入状态 2，运行 APP 程序。

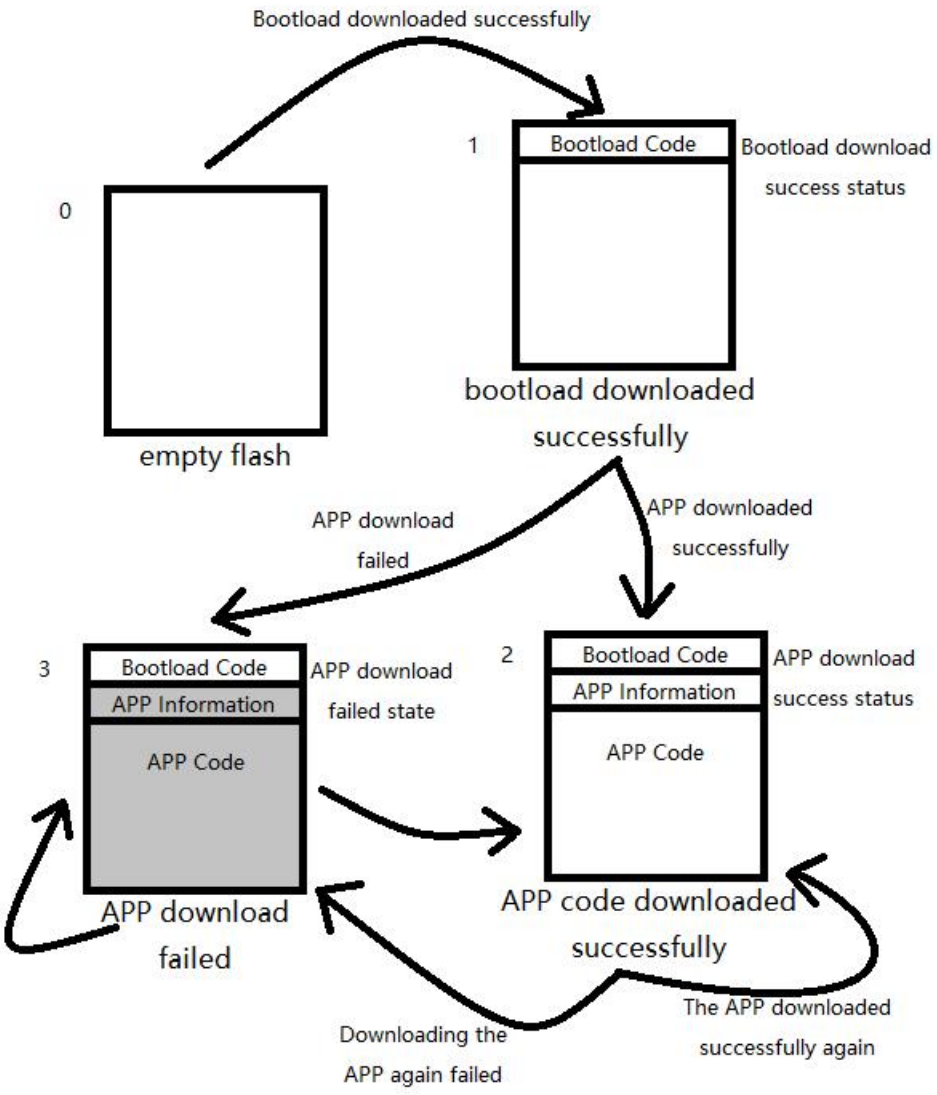


图 9

# 3.下载小实验

## 3.1 软件支持包

如下图 10 所示，文件内提供了 APP1、APP2、bootload、TSmaster_bootload、bootload 使用文档，APP1 文件和 APP2 文件是 LED 闪烁不同频率的 APP 例程，TSmaster_bootload 文件是配置好的 TSmaster 上位机软件例程，结合 bootload 可以实现下载 APP 的功能。bootload 文件里面是 bootload 的源代码。

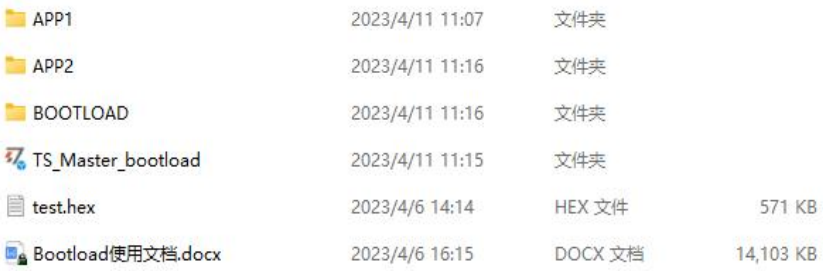


图 10

## 3.2bootload 程序下载

第一步：芯片连接下载器。

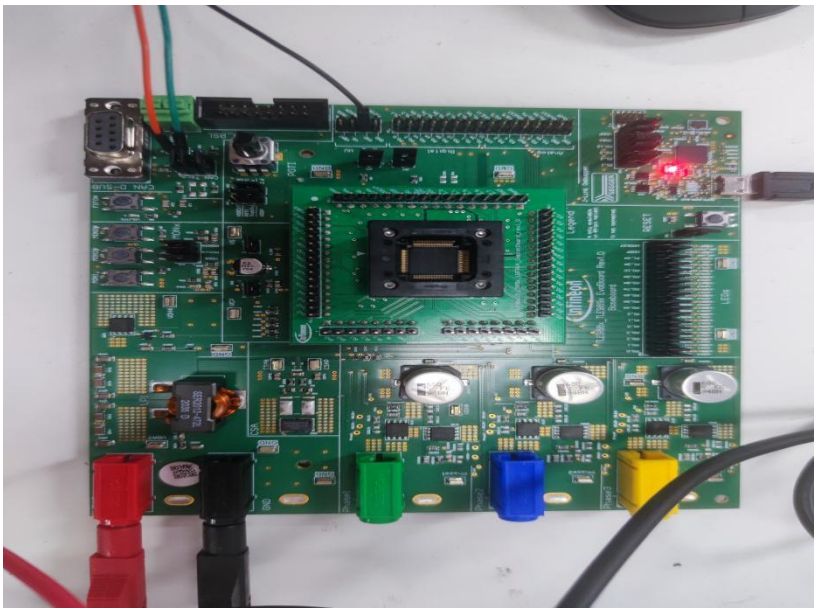

图 11

第二步：打开 bootload 文件，点击编译，再下载到单片机中。

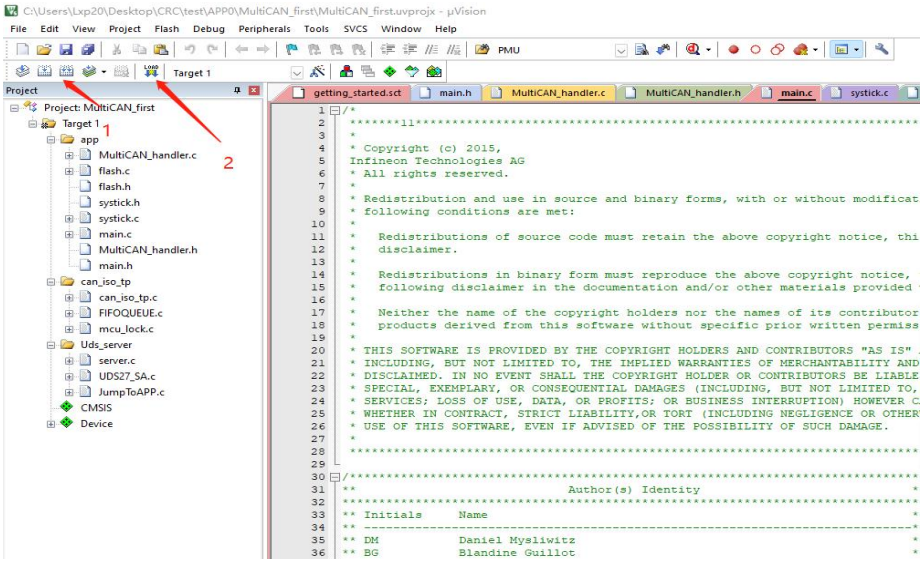


图 12

Bootload 下载成功后 P1.4 引脚的 LED 灯以 500ms 为间隔闪烁。

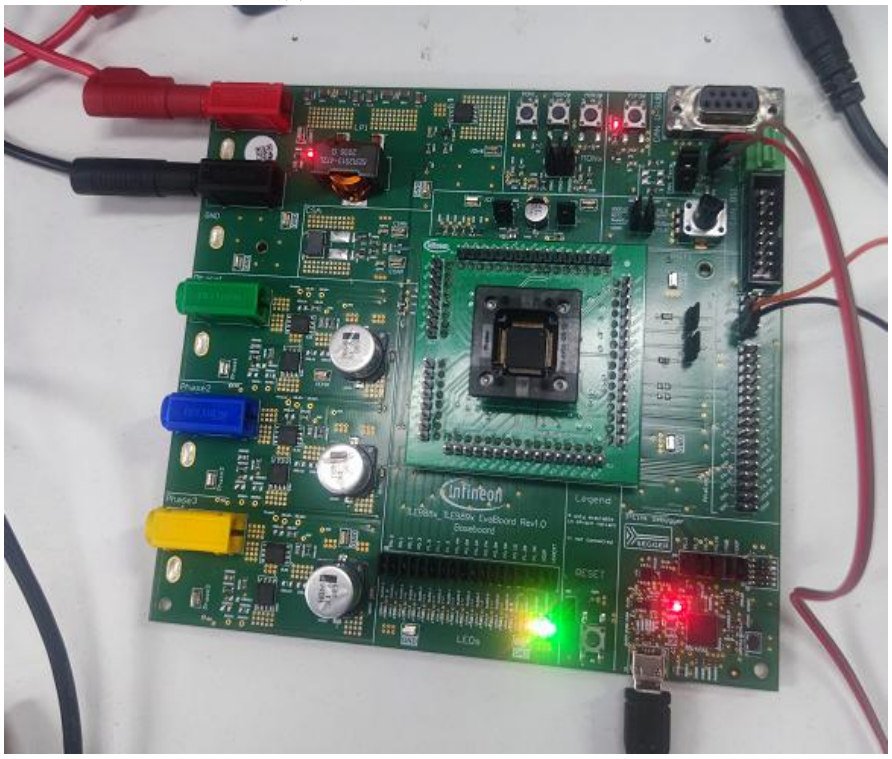

图 13

## 3.3 通过 TSMaster 下载 APP 程序

第一步：将同星 TC1012P 的 CANFD 通道连接单片机的 CANFD 通道。


图 14

第二步：打开 boot_TSmaster 文件，点击硬件—>道选择，选择 CAN，配置应用程序通道数，最后将硬件通道选择配置成 TOSUN TC1014 CANFD 通道 1。

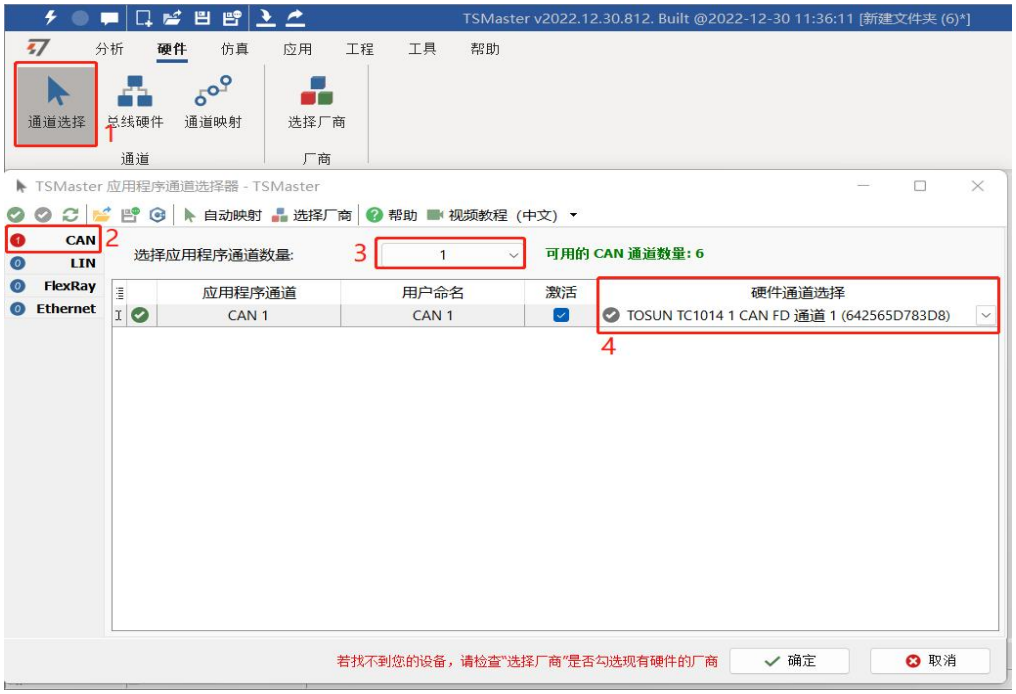


图 15

第三步：点击应用—>诊断模块—>基本诊断配置—>343637 下载文件—>文件路径加载需要下载的 hex 文件(例如 APP_LED2.hex)。

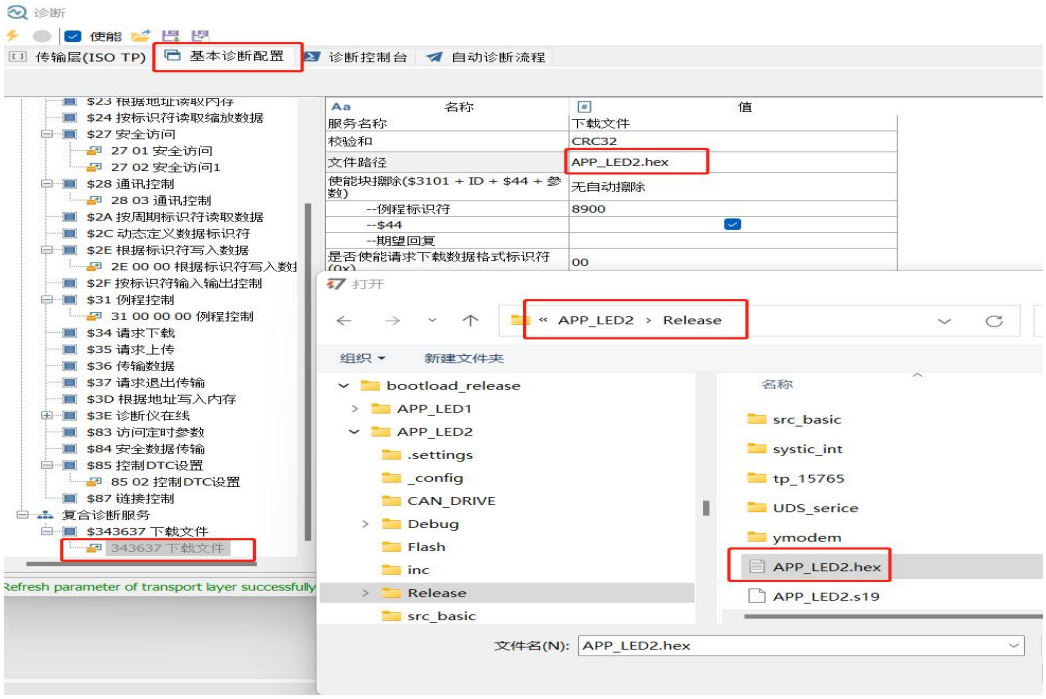


图 16

第四步：点击自动诊断模块—>下载文件—>343637 下载文件—>下载（标号 4），便可实现 APP_LED2.hex 下载到单片机内。

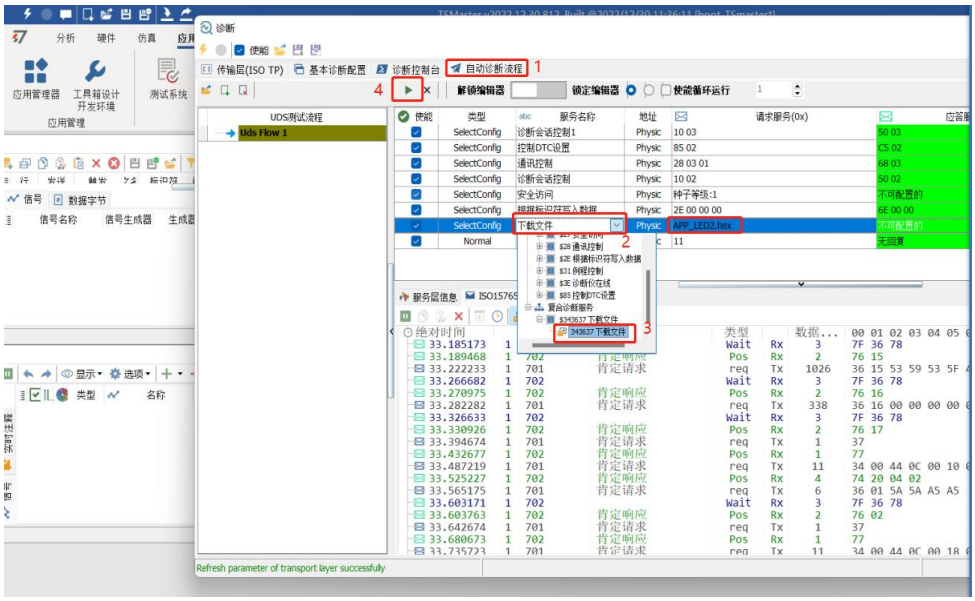


图 17

APP 下载成功后会改变 P. 13 和 P. 14 引脚 LED 灯闪烁的频率。

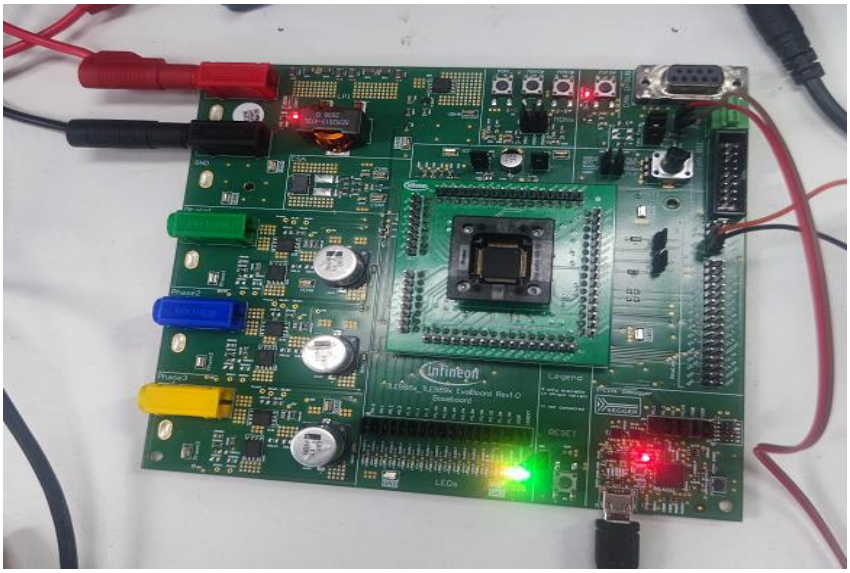

图 18

# 4.TSMaster 下载 APP 配置

## 4.1 打开诊断模块

第一步：打开 TSMaster 的诊断模块。

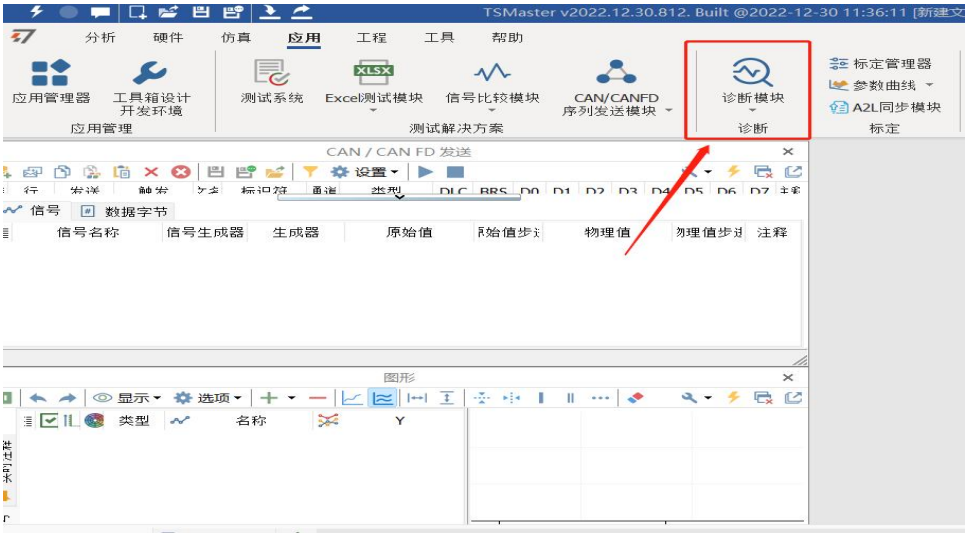


图 19

## 4.2 配置诊断传输层参数

第一步：点击传输层（ISO TP）按键。第二步：进入诊断传输层页面。第三步：将总线类型配置成 CAN/CANFD;通道根据需求进行配置（Channel1）;请求 ID 配置成 0x701;请求 ID 类型配置成 Standard;应答 ID 配置成 0x755;应答 ID 类型配置成 Standard;功能 ID 配置成 0x7DD；功能 ID 类型配置成 Standard;填充字节可以任意配置；FC 最大 DLC 配置成[15]64Bytes;FD 可变波特率可以自行勾选（勾选可以加快传输速率）。


图 20

## 4.3 配置诊断服务层参数

第一步：点击传输层（ISO TP）按键。第二步：进入诊断服务层页面。第三步：配置 P2Time(根据需求配置)；第四步配置诊断仪在线参数（根据需求配置）。第五步配置 27 服务的种子密钥。

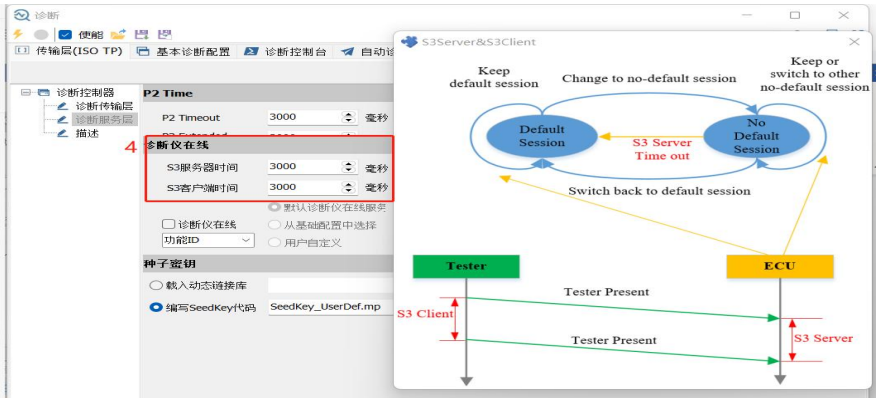


图 21

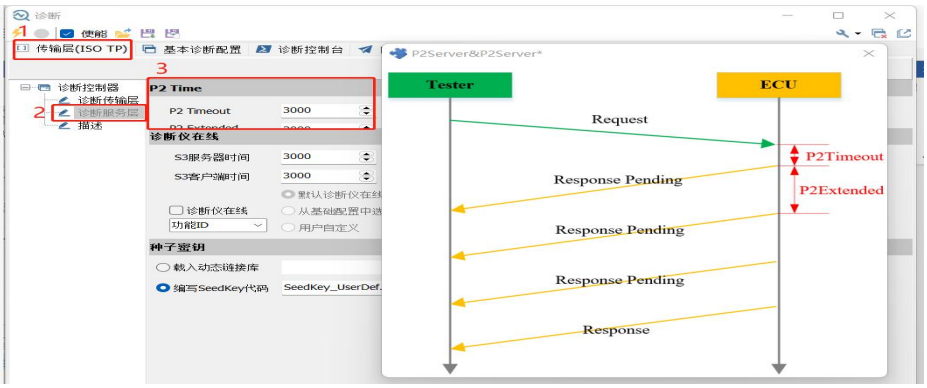


图 22

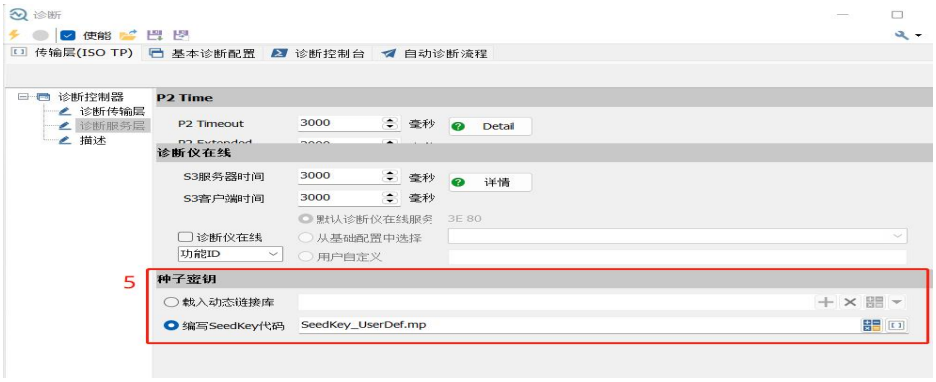


图 23

## 4.4 新建诊断服务（以 27 服务为例）

第一步进入基本配置页面；第二步：右击 27 安全访问新建一个 27 服务；第三步：修改安全反访问类型。

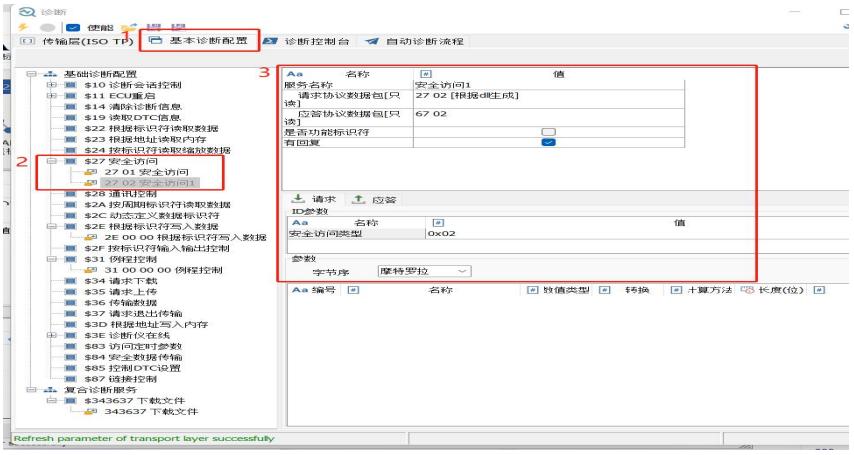


图 24

当上位机发起 27 01 请求时下位机积极响应返回 67 01+seed（如下图红色框），上位机得到 seed 后生成 key（如下图蓝色的框）发起 27 02+key 请求，下位机得到 key 后对 key 进行进行校验判断 key 是否正确，如果校验通过上上位机发送积极响应 67 02。

| | | | | | | |
|---|---|---|---|---|---|---|
| 701 | 肯定请求 | req | Tx | 2 | 27 01 | |
| 702 | 肯定响应 | Pos | Rx | 6 | 67 01 | 40 40 09 00 |
| 701 | 肯定请求 | req | Tx | 6 | 27 02 | 40 40 09 00 |
| 702 | 肯定响应 | Pos | Rx | 2 | 67 02 | |

图 25

Key 的生成规则可以同过修改 1.3 章中步骤 5 种子密钥（载入动态链接库或者 seedkey 编写代码）进行修改。

```
s32 SeedAndKey_Type2(u32 ASeed, u32* AKey) {
u8 i,length=4;
u32 key = 0xffffffff;
u8 buffer[4]={0};
buffer[0]=(u8)ASeed;
buffer[1]=(u8)(ASeed>>8);
buffer[2]=(u8)(ASeed>>16);
buffer[3]=(u8)(ASeed>>24);
while(length--)
   {
       key ^= (u32)(buffer[length]) << 24;
       for (i = 0; i < 8; ++i)
       {
           if ( key & 0x80000000 )
               key = (key << 1) ^ 0x04C11DB7;
           else
               key <<= 1;
       }
   }
*AKey=key;
return 0;
```

图 26

## 4.5 $343637 下载文件介绍

第一步新建一个 343637 下载服务。第二步在文件路径中加载需要下载的*.hex 文件;使能块擦除配置成无自动擦除；例程标识符号配置成 8900；传输退出命令配置成无校验（$37）;使能块校验配置成不做块校验。蓝色框内是加载的*.hex 文件的一些信息（下载地址要大于 0x12002000）。

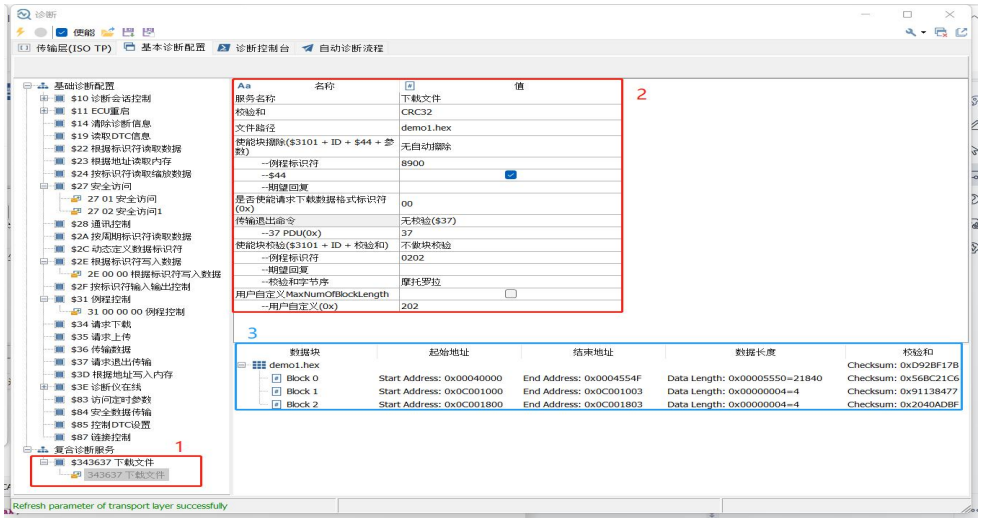


图 27

## 4.6 下载文件介绍

第一步进入自动诊断流程界面。第二步新建 UDS FLOW。第三步加载 UDS 服务请求流程（请求下载流程要符合 UDS 规范）。第四步点击下载按键开始下载*.hex 文件。蓝色框内会反馈服务请求和服务响应情况。

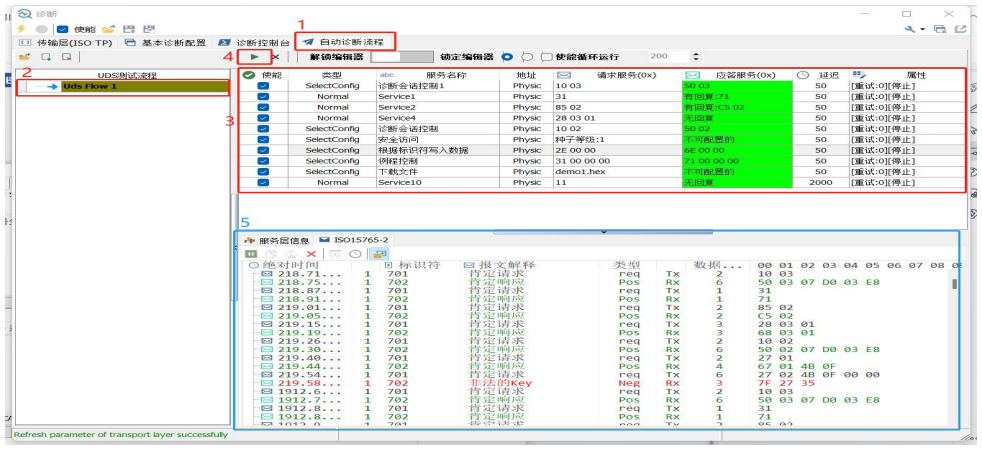


图 28

# 5.bootload 介绍

先将 bootload 程序下载到单片机中，然后 bootload 配合 TSMster 上位机可以通过 CANFD 实现下载 APP 的目的。

## 5.1bootload 目录结构如下

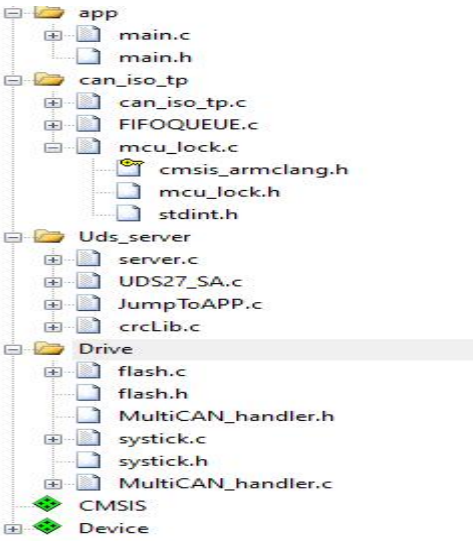


图 29

## 5.2 如何增加或删减 bootload 中 UDS 服务

通过修改 UDS_serice 目录下的 server.c 文件中的服务代码增减 UDS 服务。

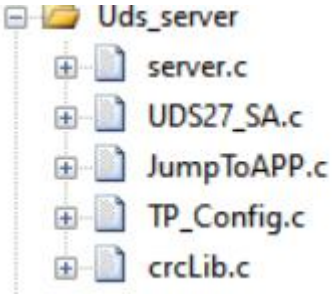


图 30

UDS 服务接口函数如下：uint8_t udsServer_requestProcess(const uint8_t payload[],uint32_t size)可以通过调用该函数实现 UDS 服务的响应。payload[]参数是指向接收数据 buffer 的指针，size 参数是接收到数据的大小。

可以通过增加或删减函数内部的 UDS 服务 ID 和 UDS 服务函数实现服务增减的目的，如下图：

```
    switch(ServerData.sid)
    {
      case UDS_SID_DiagnosticSessionControl:
      UDS_Serice_DiagnosticSessionControl();
      break;

      case UDS_SID_EcuReset:
      UDS_Serice_EcuReset();
      break;
//
//      case UDS_SID_ReadDataByIdentifier:
//      UDS_Serice_ReadDataByIdentifier();
//      break;
//
//      case UDS_SID_READ_MEM_BY_ADDR:
//      UDS_Serice_READ_MEM_BY_ADDR();
//      break;

      case UDS_SID_SecurityAccess:
      UDS_Serice_SecurityAccess();
      break;

      case UDS_SID_CommunicationControl:
      UDS_Serice_CommunicationControl();
      break;

      case UDS_SID_WriteDataByIdentifier:
      UDS_Serice_WriteDataByIdentifier();
      break;
//
//      case UDS_SID_IO_CONTROL:
//      UDS_Serice_IO_CONTROL();
//      break;

     case UDS_SID_RoutineControl:
     UDS_Serice_RoutineControl();
     break;
//
////#if BOOTLOAD
     case UDS_SID_RequestDownload:
     UDS_Serice_RequestDownload();
     break;
```

图 31

## 5.3 如何修改 bootload 中 27 服务的产生种子的函数和解锁函数

27 服务提供 Creating_Seed(){}种子（seed）生成函数;PasswordGenerator(){}钥匙（key）生成函数;SecurityAccess_unlock(){}安全访问解锁函数;Creating_Seed(){}这个函数的作用是生成安全访问所需的种子，使用者可以通过自己的需求修改内部算法产生不同的种子,UDS_SericeAccess_Seed[Access_num]数组用来存储产生的种子，Access_num 表示种子所占的字节数。PasswordGenerator(){}这个函

数根据 Seed 生成安全解锁所需的 key;UDS_SericeAccess_Key[Access_num]变量用于存储产生的 key,Access_num 表示种 key 所占的字节数。PasswordGenerator(){}这个函数用于验证外部传入的 key 和 UDS_SericeAccess_Key[Access_num]是否一致，如果一致函数返回 0 说明安全访问解锁成功，如果不一致，函数返回 1，安全访问解锁失败。函数如下图所示：

```
uint8_t UDS_SericeAccess_Seed[ACCESS_NUM]={0};
uint8_t UDS_SericeAccess_Key[ACCESS_NUM]={0};

/**
 * @brief  Generate random seeds
 * @param
 * @return None.
 * @private
 */
void  Creating_Seed(uint8_t UDS_SericeAccess_Seednum[],uint8_t keynum)
{
    //You can modify the Seed required for production by yourself
    if(keynum==4)
    {
      UDS_SericeAccess_Seednum[0]=(uint8_t)(UDS27_RN%256);
      UDS_SericeAccess_Seednum[1]=(uint8_t)(UDS27_RN%100);
      UDS_SericeAccess_Seednum[2]=(uint8_t)(UDS27_RN%55);
      UDS_SericeAccess_Seednum[3]=(uint8_t)(UDS27_RN%8);
    }
}

/**
 * @brief  Keys are generated according to the algorithm, and the algorithm for generating keys can be changed
 * @param
 * @return None.
 * @private
 */
void  PasswordGenerator(const uint8_t UDS_SericeAccess_Seednum[],uint8_t UDS_SericeAccess_keynum[],uint8_t keynum)
{
   //You can modify the Key according to the Seed
    uint8_t i;
    uint32_t key = 0xffffffff;
    while(keynum--)
    {
       key ^= (uint32_t)(UDS_SericeAccess_Seednum[keynum]) << 24;
       for (i = 0; i < 8; ++i)
       {
           if ( key & 0x80000000 )
               key = (key << 1) ^ 0x04C11DB7;
           else
               key <<= 1;
       }
    }
   UDS_SericeAccess_keynum[0]=(uint8_t)key;
   UDS_SericeAccess_keynum[1]=(uint8_t)(key>>8);
   UDS_SericeAccess_keynum[2]=(uint8_t)(key>>16);
   UDS_SericeAccess_keynum[3]=(uint8_t)(key>>24);
}
```

```
/**
 * @brief  Check that the key is authentic
 * @param
 * @return 0:success
           1:false
 * @private
 */
uint8_t  SecurityAccess_unlock(uint8_t UDS_SericeAccess_TX[],uint8_t UDS_SericeAccess_Keynum[],uint8_t keynum)
{

  for(int i=0;i<keynum;i++)
  {
    if (UDS_SericeAccess_TX[i]==UDS_SericeAccess_Keynum[i])
    {

  }
    else
    {
      return 1;
    }
  }

  return 0;

}
```

图 32

# 6.测试报告

## 6.1 测试报告

| 测试条目 | 预期结果 | 实际结果 | 备注/说明 |
|---|---|---|---|
| （使用 TSMaster）使用上位机软件，烧写 APP 应用程序 | APP 功能正常运行 | 正常 | |
| 刷写 APP 应用程序后，可连续多次刷写（如 5 次及以上） | APP 功能正常运行 | 刷写 5 次以上，正常 | |
| bootloader 下位机软件修改 App 起始地址（在正常地址范围内） | 可正常刷写，APP 正确跳转，功能正常运行 | 正常 | 地址从 0x12002000 开始到 12040000 |
| 首次刷写 App 时，在连接阶段，对 ECU 进行掉电操作。再次上电后，可重新再次刷写。 | APP 功能正常运行 | 正常 | |
| 首次刷写 App 时，在进行 APP 程序刷写的擦除 Flash 阶段，对 ECU 进行掉电操作。再次上电后，可重新再次刷写。 | APP 可以正常刷写 | 正常 | |
| 成功刷写 App 后，再次进行 APP 程序刷写，在连接阶段，对 ECU 进行掉电操作。再次上电后，可重新再次刷写。 | APP 可以正常刷写 | 正常 | |
| 刷写过程中，将 ECU 干扰至 BusOFF 状态，ECU 可自恢复， | APP 可以正常刷写 | 正常 | |

| | | | |
|---|---|---|---|
| 正常响应刷写指令。 | | | |
| 在 APP 程序刷写的连接阶段，中断上位机通讯（包括以下几种方式：上位机软件中点击停止按钮；拔掉 CAN 通讯线；拔掉 USBCAN 接口卡）后，重新恢复通讯(包括以下几种方式:上位机软件重头开始运行；恢复 CAN 通讯线；连接 USBCAN 接口卡)，可再次正常刷写。 | APP 可以正常刷写 | 正常 | |
| 在 APP 程序刷写的擦除 Flash 阶段，中断上位机通讯（同上），重新恢复通讯（同上），可再次正常刷写。 | APP 可以正常刷写 | 正常 | |
| 刷写过程中，断开 CANH,恢复后，可再次正常刷写。 | APP 可以正常刷写 | 正常 | |
| 刷写过程中，CANH 对电源短路,恢复后，可再次正常刷写。 | APP 可以正常刷写 | 正常 | |

## 6.2 测试现象

1.人为模拟一个 test.Hex 文件，地址从 0x12002000 开始到 12040000，下载成功。


2. 人为模拟一个 test.Hex 文件，下载 5 次，无出现失败。