上海同星智能科技有限公司
Shanghai TOSUN Technology Ltd.

Bootload manual based on UDS

V0.1

文档修订历史：

日期	作者	更新内容	备注
2023-03-30	TOSUN	Creating documents.	

上海同星智能科技有限公司

March, 2023

目录

Introduction

This paper introduces a general Bootloader implementation method of the TLE989X series. Bootloader can remotely upgrade the product firmware (program) through any communication port, which solves the problem that the MCU needs to dismantle the device or professional personnel, special tools, and on-site operation. The Bootloader provided this time integrates part of UDS (14229, 15765 specifications) services with the TSMaster host computer to download APP programs through the CANFD interface.

Overview

The content of this article includes: how to download the APP program through Bootload with the host computer. TSMaster was used as the host computer of Bootloader, and UDS protocol was used to transfer APP.HEX file to MCU (the underlying communication protocol was CANFD). The Bootloader program parses the data packets transmitted from the host computer, combines the APP code packets, and writes them into the target Flash space in order. The Bootloader program will automatically quit running after the APP program in the target Flash area is started successfully. The APP starts working. The flow chart of the process is as follows Figure 1:

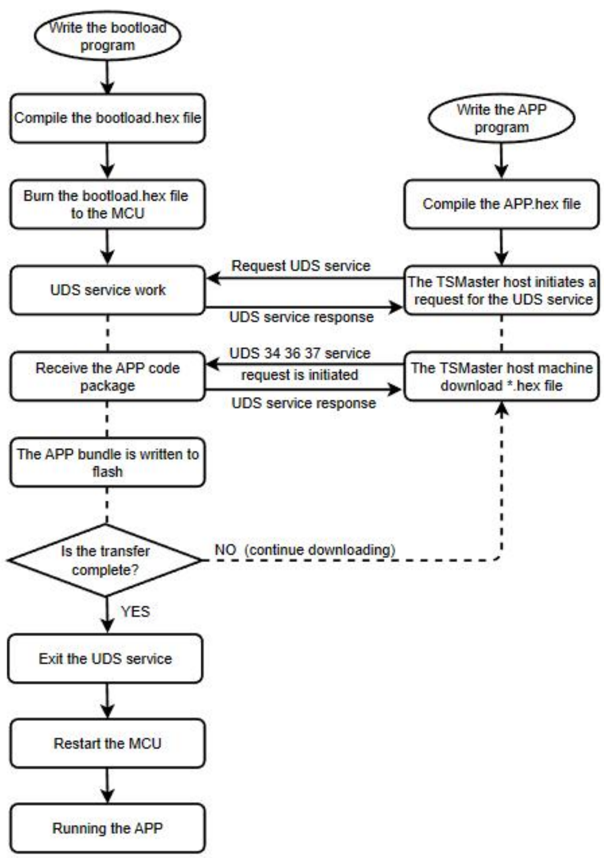


Figure 1

1. UDS

1.1 Introduction to UDS

UDS (Unified Diagnostic Services) diagnostic protocol is a diagnostic communication protocol in the environment of automotive electronic ECU, which is specified in ISO14229. At present, new cars on the market have diagnostic interfaces for out-of-vehicle diagnostics, which allows us to connect diagnostic tools to the bus system of the vehicle. Therefore, messages defined in the UDS can be sent to controllers (ECUs in the industry) that support the UDS service. In this way we can access the fault memory of the individual control units or the program to update the ECU with new firmware.

1.2UDS Common services

Category Name	SID(0x)	Diagnostic Service Name
Diagnostic and communication management function unit	10	Diagnostic Session Control
	11	ECU Reset
	27	Security Access
	28	Communication Control
	3e	Tester Present
	83	Access Timing Parameter
	84	Secured Data Transmission
	85	Control DTC Setting
	86	Response On Event
	87	Link Control
Data transfer function unit	22	Read Data By Identifier
	23	Read Memory By Address
	24	Read Scaling Data By Identifier
	2A	Read Data By Periodic Identifier
	2C	Dynamically Define Data Identifier
	2E	Write Data By Identifier
	3D	Write Memory By Address
Storage data transfer function unit	14	Clear Diagnostic Information
	19	Read DTC Information
Input and output control function unit	2F	Input Output Control By Identifier
Routine function unit	31	Routine Control
Upload and download function unit	34	Request Download
	35	Request Upload
	36	Transfer Data

	37	Request Transfer Exit
	38	Request File Transfer

Figure 2

1.3UDS download program flow

Step1：10 03 //10 Service switch to 03 extension mode

Step3: 85 02 //Off DTC(empty service, no concrete implementation)

Step4: 28 03 01 //Service gateway packet (empty service, no specific implementation)

Step5: 10 02 //10 Service switches to 02 programming session

Step6: 27 01 //27 Service, unlocked, security verified.

Step7: 27 02

Step8: 2e 00 00

Step9: 31 00 00

Step10:(34、36、37）server //Downloading the APP.

Step11:11 //ECU reset.

2. Introduction of Infineon TLE989X MCU Board

2.1Infineon TLE989X series microcontroller storage space mapping

The storage space of Infineon TLE989X series microcontroller is designed to be arranged according to linear address. The benefit of this is that RAM, ROM, Flash and register addressing is more convenient and intuitive. The physical address range for IROM1 is from 0x11000000 to 0x11006000, and for IROM2 is from 0x12002000 to 0x12040000,IRAM1 has the physical address range 0x18000000 to 0x18002000, and IRAM2 has the physical address range 0x18002000 to 0x18007C00. The specific situation is shown in Figure 3 below.

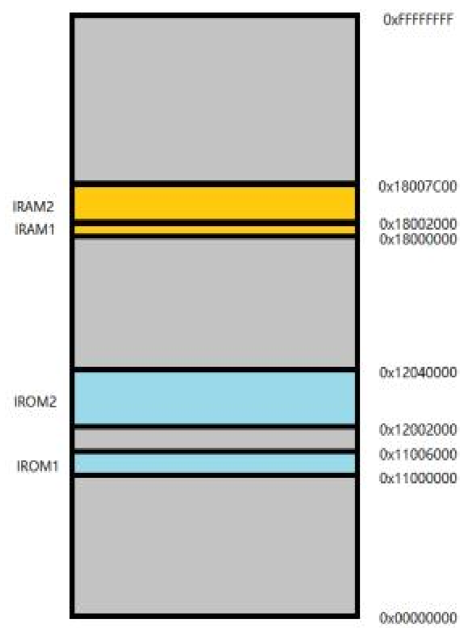


Figure 3

2.2The code storage distribution of Infineon TLE989X series microcontroller

The Bootloader of Infineon TLE989X series microcontroller is placed in the low address space of Flash starting from 0x11000000 address. After MCU is powered on, it will automatically start from Bootloader to check whether APP program code exists, and then jump to APP for execution. If it does not exist, it enters bootload and waits for TSMster to initiate UDS service request. The APP program is placed in an area behind the Flash. The APP program of this routine is stored from the address 0x1200 2000. In fact, this start address can be changed to any other start address, as long as it does not coincide with the Bootloader area and there is enough Flash space left to store the APP. The ld files for Bootload and APP are shown in Figures 4 and 5 below:

```
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)
  }
}
```

Figure 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)
  }
}
```

Figure 5

2.3Infineon TLE989X series microcontroller interrupt vector table

The interrupt vector table is essential at program startup and during the execution of interrupt service functions, which is equivalent to the "directory" of the program. In particular, 0x00000100-0x00000103 holds the stack space MSP - the value of the stack top pointer. Also, 0x0000 0104-0x0000 0107 holds the pointer to the Reset_Handler function. The following screenshot shows the contents of a partial Vector.c file：

```
__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
```

Figure 6

When the Bootloader boots the APP, it needs to use the interrupt vector table of the APP. After the APP starts, it needs to switch the Bootloader's vector table to the APP's own vector table. This process is called vector table remapping. In the conventional program, after the interrupt is generated, the hardware automatically addresses the corresponding interrupt service function entry and jumps to the interrupt function execution after the stack is pushed in the interrupt field. The jump code is as follows Figure 7:

```
/* Service Watchdog */
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 ] ) ;
```

Figure 7

2.4Download the APP process

The microcontroller is powered on and started, the CAN is initialized, and the delay is 50 milliseconds to wait for whether to enter the Bootload mode. If the Bootload mode is not entered, check whether there is an APP program at the address 0x12002000. If the APP exists and there is no UDS service request, the MCU restarts and jumps to the address 0x12002000 to execute the APP program. If the APP program is not detected and there is no UDS service request, the MCU will enter the Bootload mode to respond to the UDS service (such as 24, 26, 27 download service) request initiated by the host computer. Until the host computer initiates the 11 service (microcontroller reset) request, the microcontroller is reset to detect whether the APP code is downloaded. If the APP download is completed, the bootload mode will exit and the APP code program will be executed at 0x12002000. If the download is not completed, the current state will still be bootload mode. If the APP program has been downloaded before, the APP program can be directly re-downloaded in APP mode without power down and restart. If there is an error in the APP, it can power off and restart. Before the 50ms delay ends, it can send the instruction to enter bootload mode to download the APP again. The flow chart of Bootload downloading APP is as follows Figure 8:

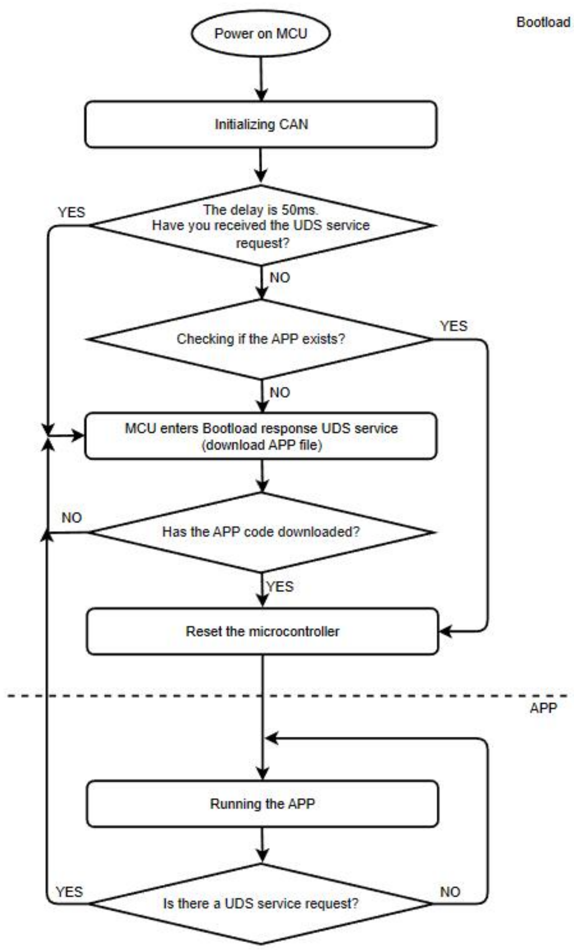


Figure 8

2.5MCU Status Introduction

Get a new chip, MCU is in state 0, then start to download bootload code, when the bootload code download success will enter state 1, bootload code start to work. Once the MUC is in state 1, you can start downloading the APP code. If the APP code download fails, it will enter state 3, at this time, the MCU is in bootload mode, and it can retry to download the

APP program. If APP is downloaded successfully, it will enter state 2, the MUC will jump from bootload mode to APP mode, and the APP will start to work. If the APP code needs to be re-downloaded, if the process of downloading APP is interrupted or an error occurs, the MCU will re-enter state 3 and wait for the APP code to be re-downloaded until the re-download of APP is successful, then enter state 2 and run the APP program.

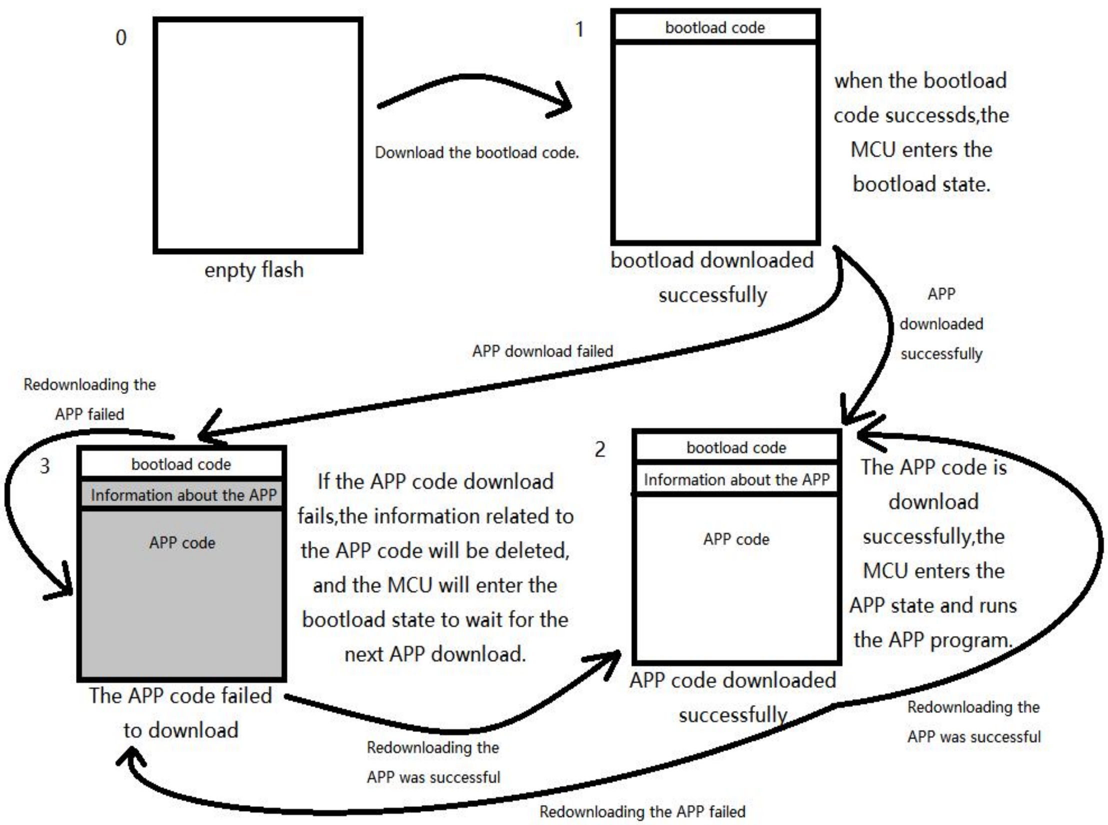


Figure 9

3.Download the experiment

3.1Software support package

As shown in Figure 10 below, APP1, APP2, bootload, TSmaster_bootload, and bootload usage documents are provided in the file. APP1 file and APP2 file are APP routines for LED flashing at different frequencies. TSmaster_bootload file is the configured TSmaster host computer software routine, combined with bootload can realize the function of downloading APP. The bootload file contains the bootload source code.

APP1	2023/4/11 11:07	文件夹	
APP2	2023/4/11 11:16	文件夹	
BOOTLOAD	2023/4/11 11:16	文件夹	
TS_Master_bootload	2023/4/11 11:15	文件夹	
test.hex	2023/4/6 14:14	HEX 文件	571 KB
Bootload使用文档.docx	2023/4/6 16:15	DOCX 文档	14,103 KB

Figure 10

3.2bootload program download

Step 1: The chip is connected to the downloader.

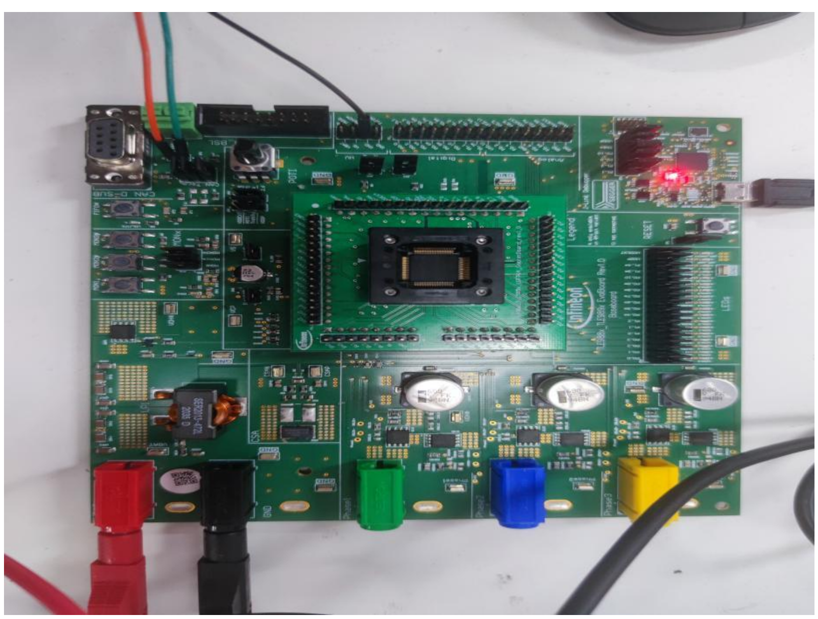


Figure 11

Step 2: Open the bootload file, click compile, and then download to the MCU.

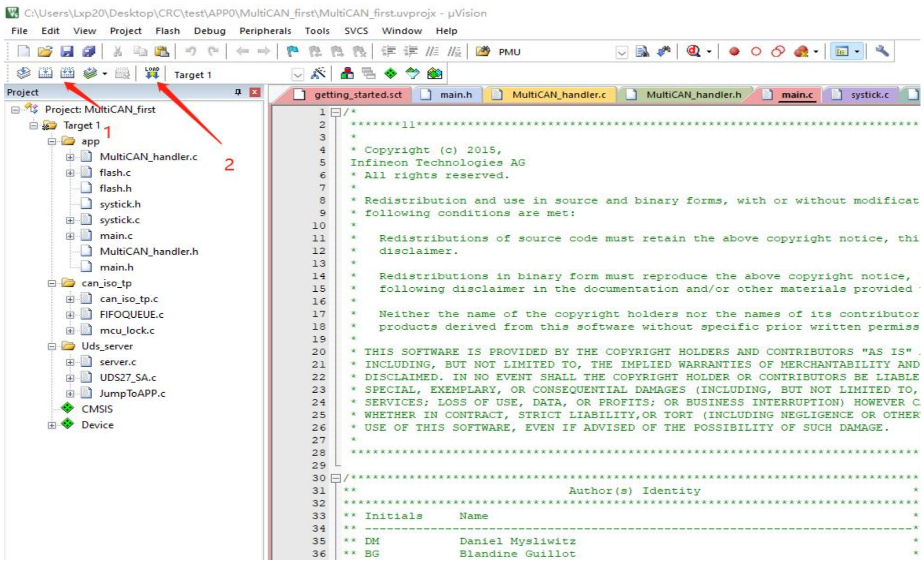


Figure 12

After Bootload download is successful, the LED light of P1.4 pin flashes at 500ms interval.

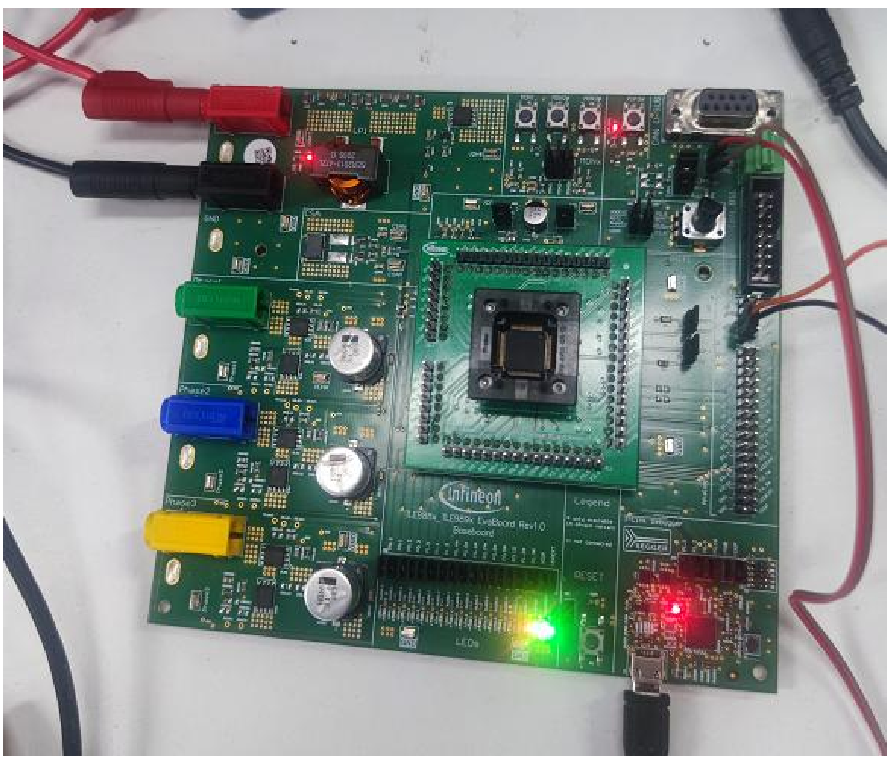


Figure 13

3.3Download the APP via TSMaster

Step 1: Connect the CANFD channel of the same star TC1012P to the CANFD channel of the MCU.

Figure 14

Step 2: Open boot_TSmaster file, click Hardware - > Channel selection, select CAN, configure the number of channels in the application, and finally configure the hardware channel selection to TOSUN TC1014 CANFD channel 1.

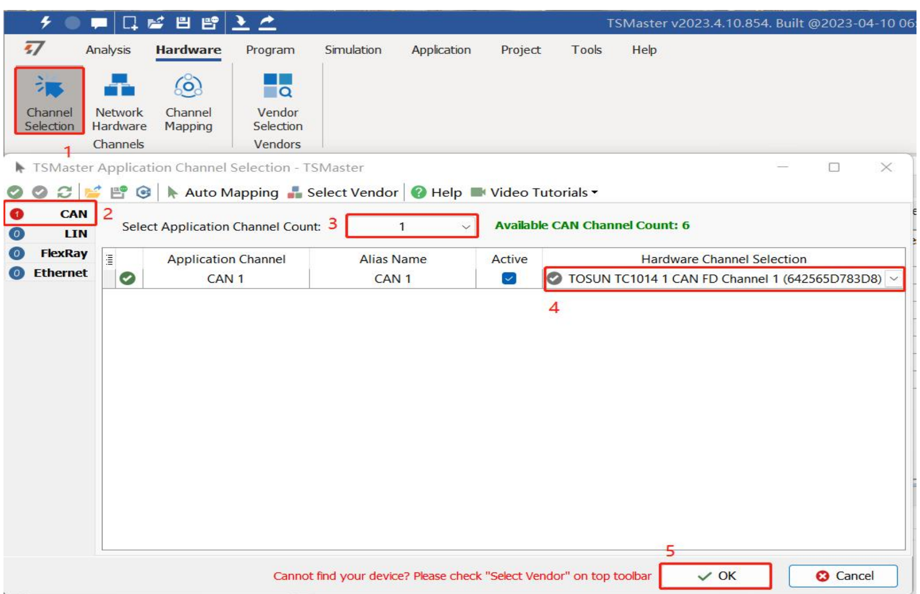


Figure 15

Step 3: Go to App - > Diagnostic Module - > Basic Diagnostic Configuration - >343637 Download File - > File Path to load the hex file you want to download (e.g. App_lef2.hex).

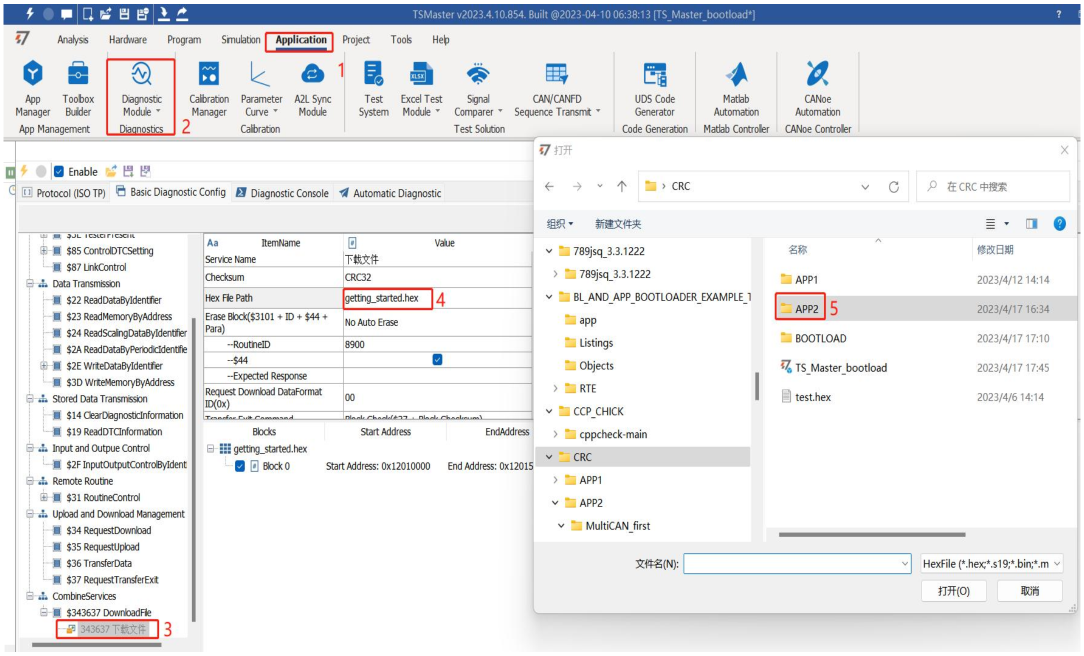


Figure 16

Step 4: Click Automatic diagnosis module - > Download file - >343637 Download file - > Download (label 4), you can download APP_LED2.hex to the MCU.

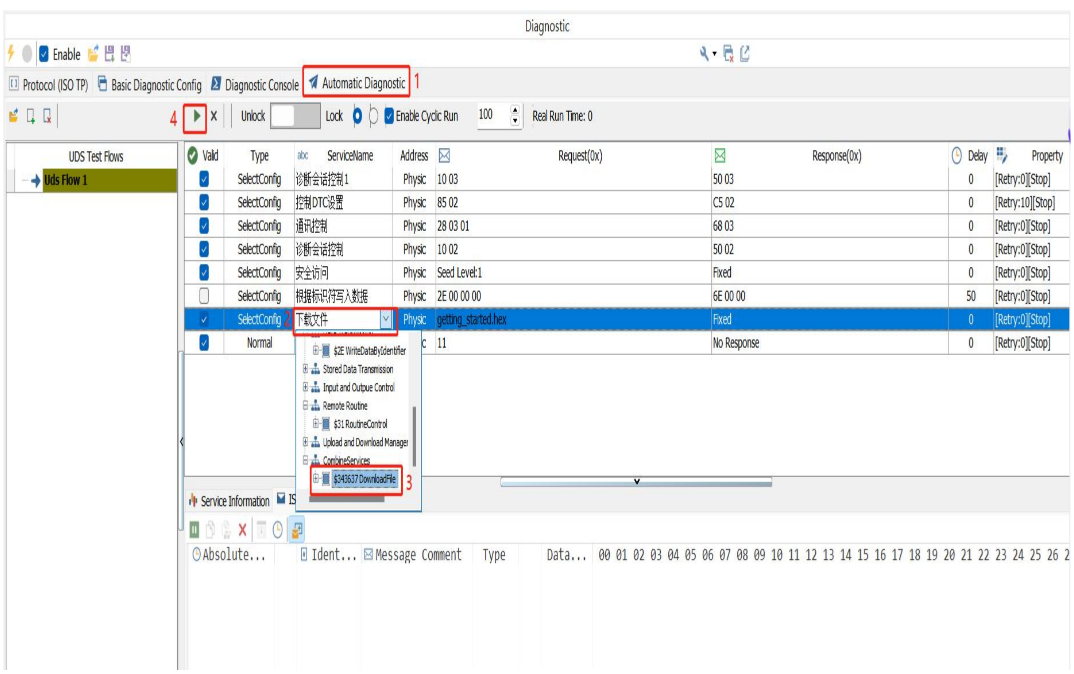


Figure 17

After successful APP download, it will change the frequency of P.13 and P.14 pin LED lights flashing.

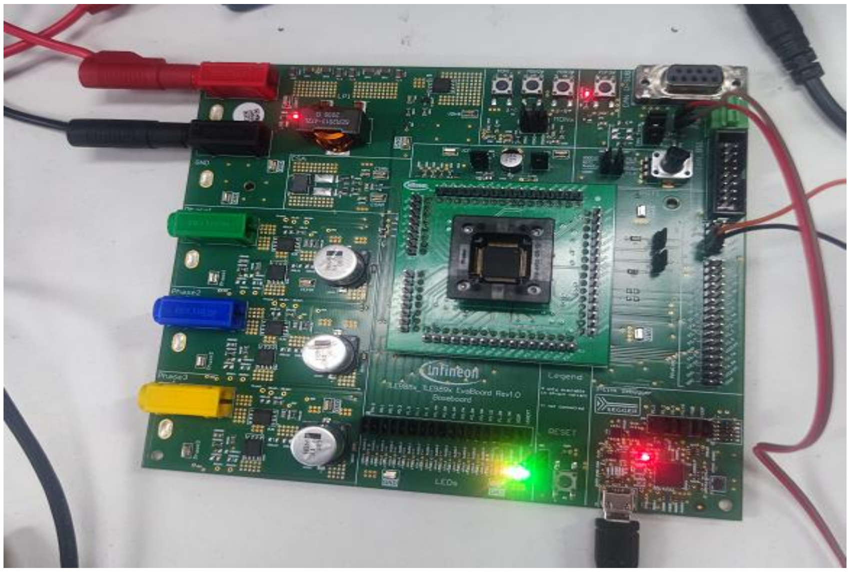


Figure 18

4.TSMaster downloads the APP configuration

4.1Open the diagnostic module operation

Step 1: Open TSMaster's diagnostic module.

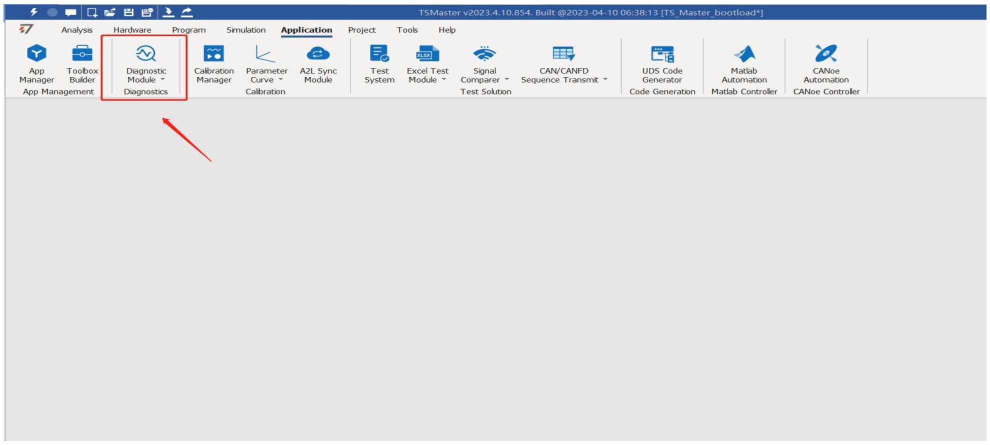


Figure 19

4.2Configure diagnostic transport layer parameters

Step 1: Click the Transport Layer (ISO TP) button. Step 2: Go to the Diagnostic Transport layer page. Step 3: Configure the bus type as CAN/CANFD; Channels are configured on demand (Channel1) The request ID is configured to 0x755. The request ID type is set to Standard. The reply ID is configured to 0x7FF. The reply ID type is configured to be Standard. Function ID is configured to 0x7DD. The function ID type is set to Standard. Padding bytes can be configured arbitrarily. The maximum DLC of FC is set to [15]64Bytes. FD variable baud rate can be checked (checked to speed up the transmission rate).

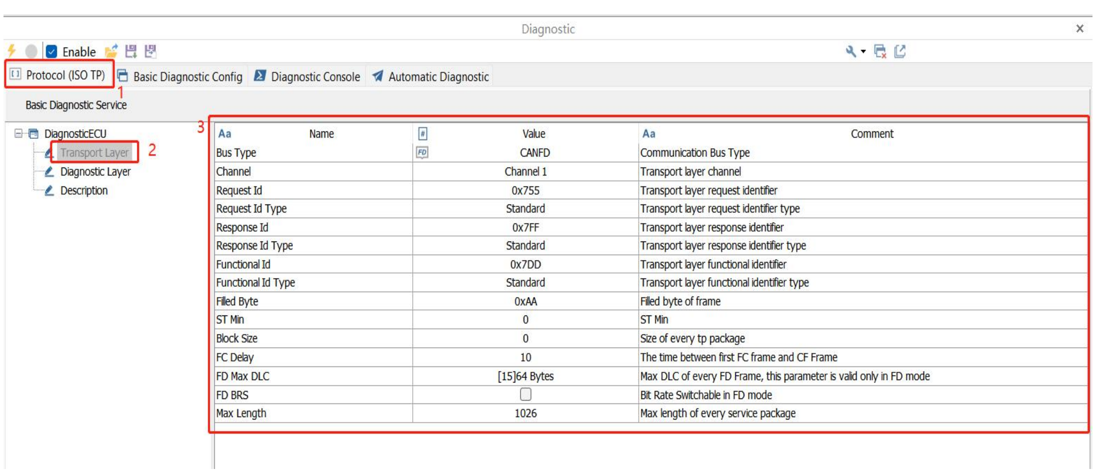


Figure 20

4.3Configure the diagnostic service layer parameters

Step 1: Click the Transport Layer (ISO TP) button. Step 2: Go to the diagnostic service layer page. Step 3: Configure P2Time(on demand) Step 4 Configure the online parameters of the diagnostic instrument (configure according to requirements). Step 5 config the seed key for the 27 service.

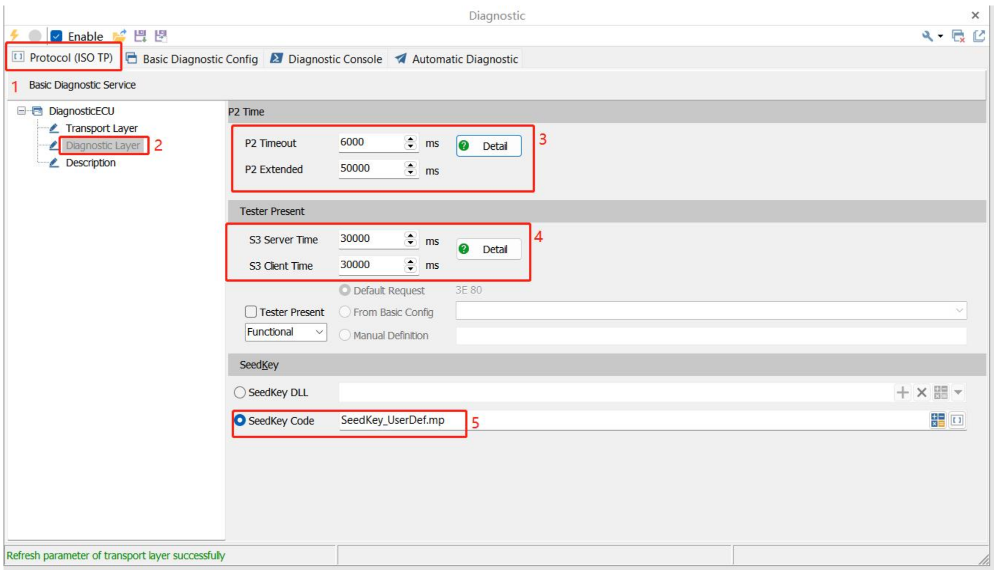


Figure 21

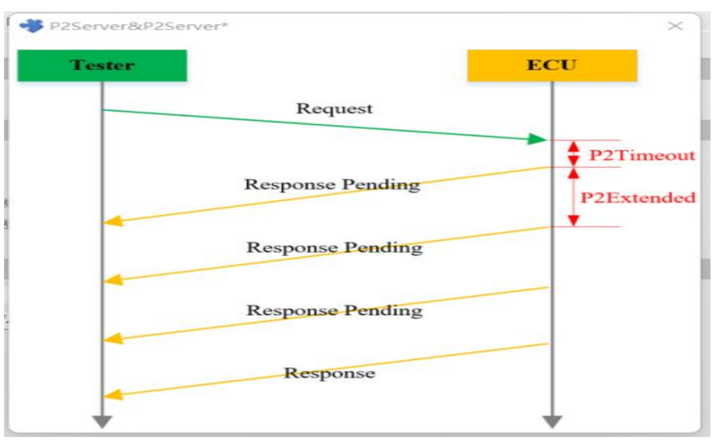


Figure 22

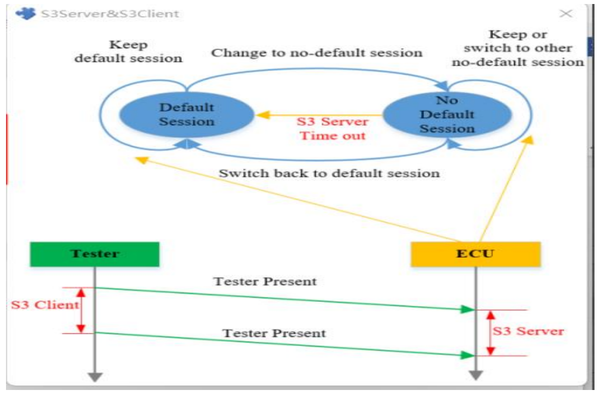


Figure 23

4.4Create a new diagnostic service (consider the 27 service)

The first step is to go to the basic configuration page. Step 2: Right-click 27 Secure access to create a new 27 service; Step 3: Modify the security anti-access type.

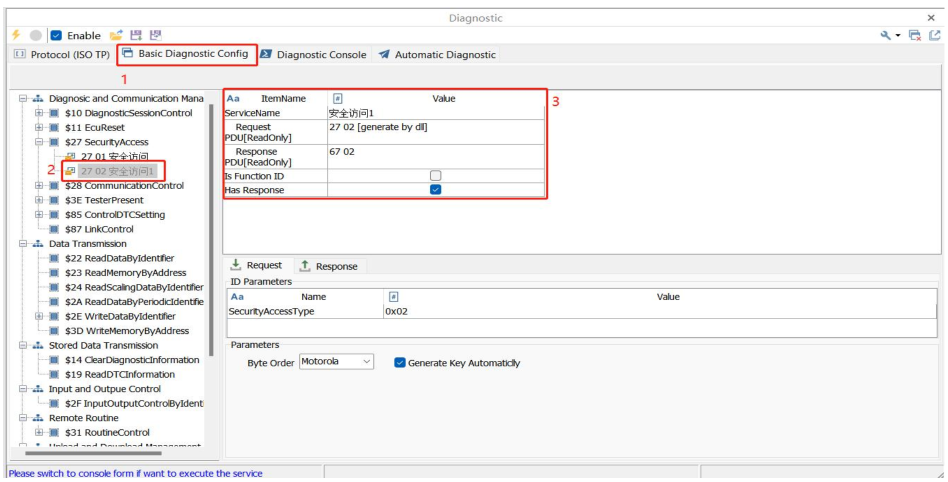


Figure 24

When the host computer initiates 27 01 request, the lower computer responds positively and returns 67 01+seed (red box in the following picture). After the host computer obtains the seed, it generates the key (blue box in the following picture). It initiates 27 02+key request. If the verification sends a positive response through the upper computer 67 02。

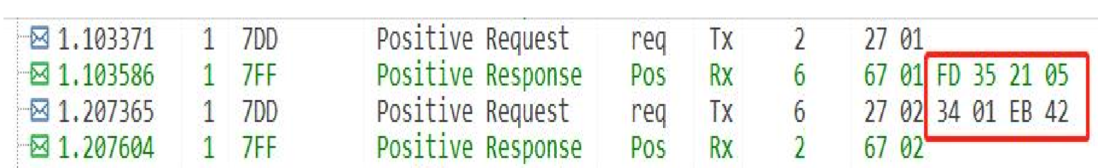


Figure 25

The Key generation rule can be modified by modifying the seedkey in step 5 of Chapter 1.3 (loading dynamic link library or writing code with seedkey).

```
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;
```

Figure 26

4.5 $34 36 37 Download the file description

The first step is to create a new 34 36 37 download service. The second step is to load the *.hex file in the file path. Enable block erasure to be configured without automatic erasure. The routine identification symbol is configured to 8900. The transfer exit command is configured without validation ($37). Enable block validation is configured to do no block validation. The blue box contains some information about the loaded *.hex file (the download address should be greater than 0x12002000).

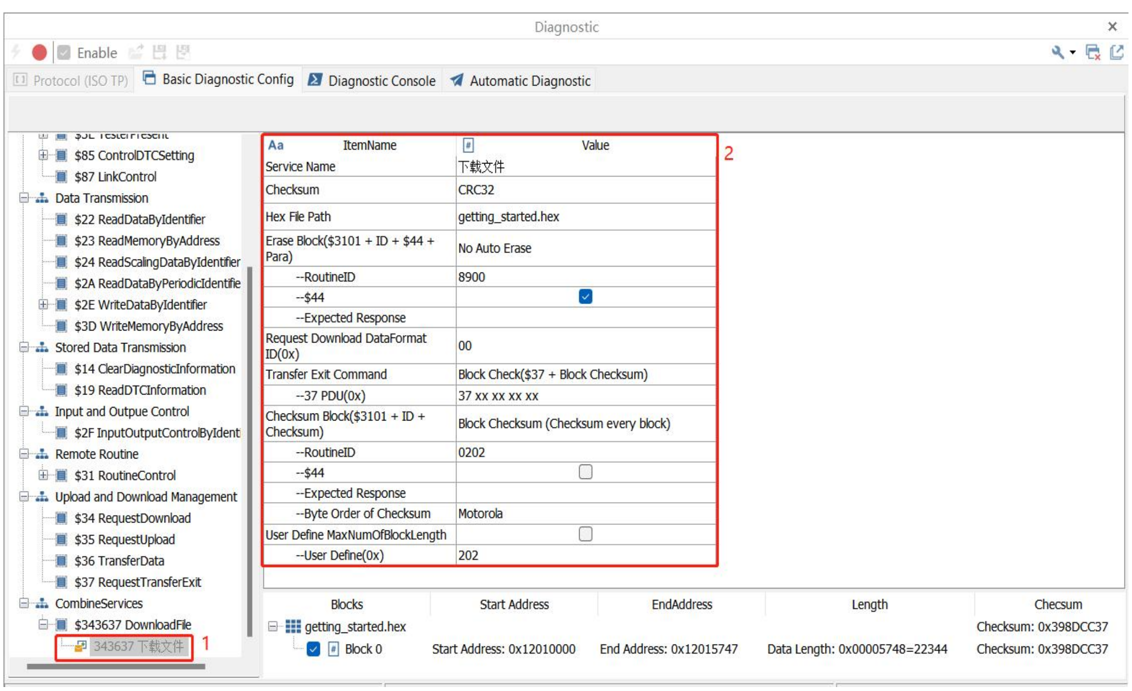


Figure 27

4.6Download the file description

The first step is to enter the automatic diagnosis process interface. The second step is to create a UDS FLOW. The third step is to load the UDS service request process (the request download process should conform to the UDS specification). Step 4 Click the Download button to start downloading the *.hex file. The blue box gives feedback on service requests and service responses.

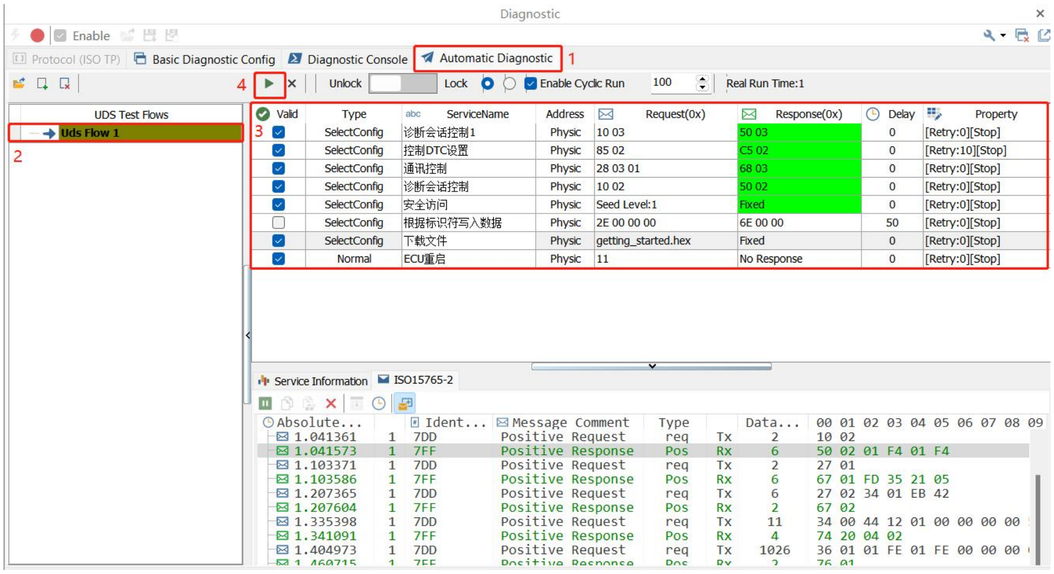


Figure 28

5.Introduction to bootload

The bootload program is downloaded to the MCU, and then bootload with the TSMster host computer can realize the purpose of downloading APP through CANFD。

5.1The bootload directory is structured as follows

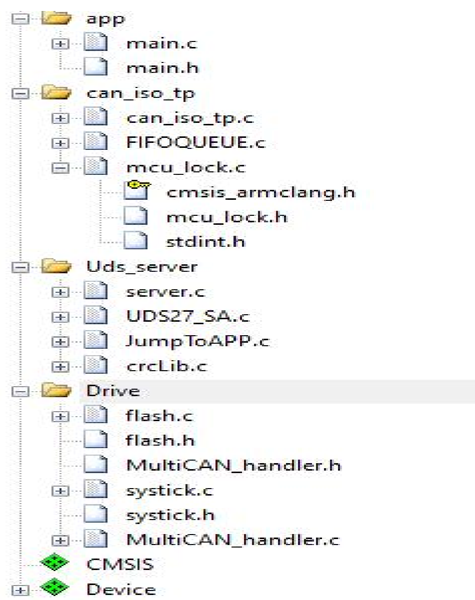


Figure 29

5.2How to add or delete the UDS service in bootload

Add or remove the UDS service by changing the service code in the server.c file in the UDS_serice directory.

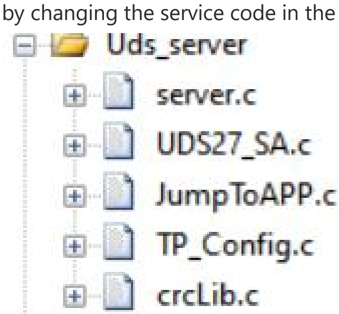


Figure 30

The UDS service interface function is as follows: uint8_t udsServer_requestProcess(const uint8_t payload[],uint32_t size) can be called to implement the UDS service response. The payload[] parameter is a pointer to the buffer of the received data, and the size parameter is the size of the received data.The UDS service ID and UDS service function inside server.c can be added or deleted to achieve the purpose of service increase or decrease, as shown in Figure 31 below:

```
*/
void UdsServerRequestProcess(const  uint8_t * payload,uint32_t size)
{

    ServerData.rxMsgLength= size;
    ServerData.sid=payload[0];
    if(ServerData.rxMsgLength>0)
    {
        for (uint32_t i = 0;i<ServerData.rxMsgLength;i++)
        {
          ServerData.rxMsgData[i]=payload[i];

        }
    }
    else
    {
    }

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

Figure 31

5.3Modify the functions and unlocking functions of the 27 service generation seed

The service provides the Creating_Seed(){} seed generation function; PasswordGenerator(){} key generator function; SecurityAccess_unlock(){} Secure access to the unlock function; The Creating_Seed(){} function is used to generate the seed needed for secure access. Users can modify the internal algorithm to generate different seeds according to their own requirements. The UDS_SericeAccess_Seed[Access_num] array is used to store the generated seed. Access_num indicates the number of bytes occupied by the seed. PasswordGenerator(){} This generates the key needed to secure the unlock from the Seed; The variable UDS_SericeAccess_Key[Access_num] is used to store the generated key, and Access_num indicates the number of bytes in the key. PasswordGenerator(){} This function validates keys and UDS_ passed in from outside Whether SericeAccess_Key[Access_num] is consistent. If the consistent function returns 0, the secure access unlocking is successful. If not, the function returns 1, the secure access unlocking is failed. The function looks like this：

```
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);
    }
}
```

```c
/**
 * @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);
}
```

```c
/**
 * @brief  Check that the key is authentic
 * @param
 * @return 0:success
           1:false
 * @private
 */
uint8_t  SecurityAccess_unlock(uint8_t UDS_SericeAccess_RX[],uint8_t UDS_SericeAccess_keynum[],uint8_t keynum)
{
    for(int i=0;i<keynum;i++)
    {
       if (UDS_SericeAccess_RX[i]==UDS_SericeAccess_keynum[i])
       {

       }
       else
       {
         return 1;
       }
    }
    return 0;
}
```

```c
/**
 * @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;

}
```

Figure 32

6. test report

6.1test report

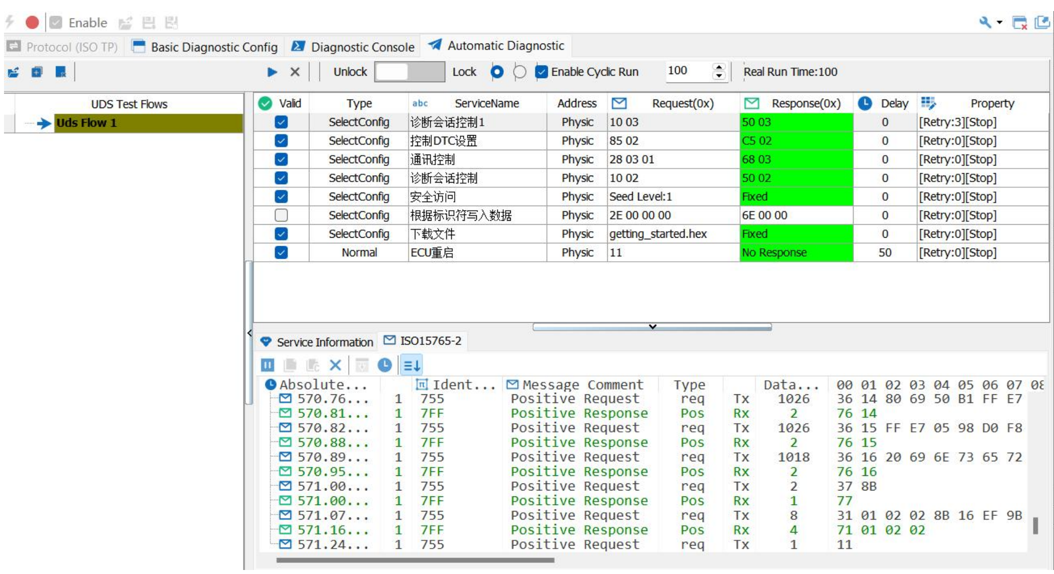
Test entries	expected result	actual result	Remark/explanation
(using TSMaster) Use the host computer software to burn the APP application	The APP functions normally	OK	
After the APP has been flushed, it can be flushed multiple times in a row (e.g., 5 or more times)	The APP functions normally	Flush more than 5 times, OK	
Bootloader software modifies the start address of App (within the normal address range)	It can be brushed normally, the APP jumps correctly, and the function runs normally	OK	Addresses start from 0x12002000 to 12040000
When the App is flushed for the first time, power down operation is performed on the ECU during the connection phase. After power on again, you can re-brush again.	The APP functions normally	OK	
When the App is flushed for the first time, power down operation is performed on the ECU in the erase Flash stage of APP program flushing.	The APP can be flushed normally	OK	

After power on again, you can re-brush again.			
After the App is successfully flushed, the APP program is flushed again. In the connection phase, the power down operation is performed on the ECU. After power on again, you can re-brush again.	The APP can be flushed normally	OK	
In the flush process, the ECU is disturbed to the BusOFF state, and the ECU can recover itself and respond to the flush instructions normally.	The APP can be flushed normally	OK	
In the connection phase of the APP program flush, interrupt the host computer communication (including the following ways: click the stop button in the host computer software; Unplug the CAN communication line; Unplug the USBCAN interface card) and resume communication (including the following ways: the host computer software starts running again; Restore the CAN communication line; Connect the USBCAN interface card), can again normal flush.	The APP can be flushed normally	OK	
In the Flash erase phase of the APP program flush, interrupt the host computer communication	The APP can be flushed normally	OK	

(ditto above) and resume the communication (ditto above), and the normal flush can be performed again.			
During the flush process, CANH is disconnected, and after recovery, it can be flushed normally again.	The APP can be flushed normally	OK	
During the flush process, the CANH short-circuits the power supply, and after recovery, it can flush normally again.	The APP can be flushed normally	OK	

6.2Test phenomenon

1.A mock test.hex file was downloaded 5 times without failure.



2. Artificially simulate a test.Hex file from 0x12002000 to 12040000.