PAPER • OPEN ACCESS

UDS in CAN flash programming

UDS in CAN flash programming

Li Du[1,*], Peng Xie[2] ,Bing Zhou[3] ,Yongyan Yu[4] ,Juan Wan[5] ,Haixia Hu[6] and Lianghao Cui [7]

[1-7] Dongfeng Motor Technology Center,WuHan,Chi*na;*

*Corresponding author e-mail: duli@dfmc.com.cn

Abstract. Flash bootloader is the most important part in the ECU platform software. It is the base function of the product in developing and after-sale, supporting self-flash. CAN bus is the main network in vehicle, which connect the whole network together. Thus, flash programming through CAN bus, bring us convenient in whole vehicle flashing.

1. Introduction

With the development of CAN bus technology, ISO organization issued relevant international standards, such as ISO 15765, ISO 14229, ISO 15031, etc. At present, vehicle factories have gradually begun to develop flashing based on CAN bus for meeting vehicle ECU program upgrade requirements, through the connection of vehicle OBD port[1][2].

This paper briefly introduced the technical usage in flashing and takes Freescale S12 series chip for an example that the use of UDS in BootLoader .

2. Introduction of ECU flashing

2.1 Requirement of flasing

Software problems ,which caused the vehicle running abnormally, were found after supplier has delivered electronic products and the whole vehicle was loaded. In traditional, suppliers often choose to replace abnormal ECU and reinstalled new one. This approach was inconvenient and took lots of time because suppliers were not located in the city where OEM located in.

There are dozens of ECUs in the whole vehicle. If each ECU have requirement of software updates, it will require a lot of manpower and material resources according to the above approach. Therefore, OEM proposed flashing requirement, which is to update the program of the ECU components and setting up new standards in the absence of any hardware changes to the vehicle.

2.2 UDS and CAN bus

Obviously, it was the easiest way to update programs from OBD port through CAN bus to fulfill the above requirements. In recent years, with the development of CAN bus, vehicle manufactures have gradually formulated a set of strategies to regulate the operation of the vehicle network. SAE tailored and supplemented the standard 7-tier communication model defined by OSI to form the CAN network model.

UDS get a set of standard diagnostic services of CAN bus between ECU and diagnostic devices constituted by SAE[3]. SAE provides the content and port mode of diagnostic services in ISO14229. As shown in Table 1, there is a correspondence between UDS and OSI layer.

Table 1. Correspondence between UDS and OSI layering

OSI layer	Supplier
Application Layer	ISO 15765-3/ISO 14229
Presentation Layer	N/A
Session Layer	ISO 15765-3
Transport Layer	N/A
Network Layer	ISO 15765-2
Data Link Layer	ISO 11898
Physical Layer	ISO 11898

As shown in Figure 1, we designed the following software architecture to corresponding to the above model, which is divided into four layers from bottom to top.

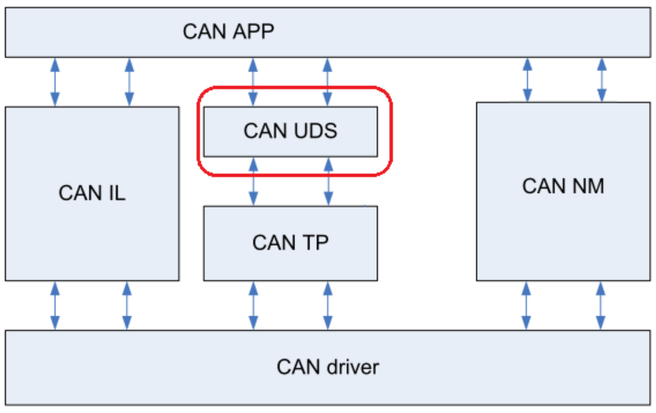

Figure 1. CAN bus software architecture

In the software architecture,the first layer was CAN driver, which obtained message content (including physical layer and data link layer) from CAN bus according to ISO 11898.

The second layer was the network layer, which is responsible for sending and receiving network data, and its key content was to process the decomposition and assembly of large data packets according ISO 15765-2[4].

The third layer was the Diagnostic Layer (UDS), which defined the commands and service of diagnosing communication standards between the diagnostic instrument and the controller.

The fourth level was the application layer, which pass data to each other's nodes through various services of UDS.

2.3 Network layer data transfer

For a UDS message, ISO14229 regulates that the longest message for a service can be 4095 bytes[4]. However, a single CAN message is limited to a maximum of 8 bytes in ISO 11898. It is necessary to distribute the message into multiple data frames to realize that long data sending to MCU controller which can only obtain 8 bytes each time. Therefore, SAE defined ISO 15765-2 to regulate the data transmission process. As shown in Figure 2, when there was a large amount of data to be sent out at the sender, it must send out some date frames through the network layer.

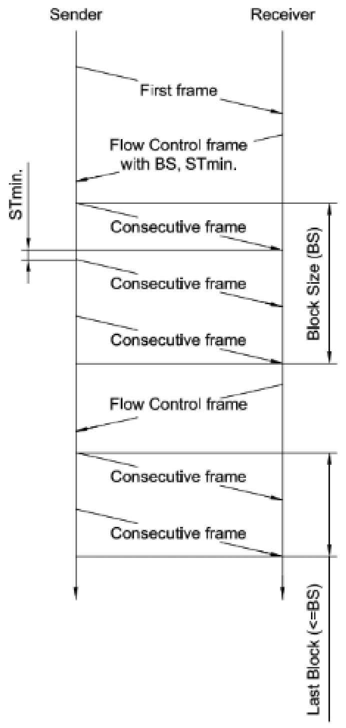

Figure 2. Date frames in network layer

The STmin parameter is the minimum interval, which represents the time interval between the two frames. The BS parameter is the frequency of persistent transmission, which indicates the maximum times of frames sent in an interaction.

By framing, UDS service can achieve the interaction among long messages to provide a basic data channel for large data transmission in the process of flashing.

2.4 UDS instruction introduction

In the previous section, it was explained that the process of forming UDS messages from ordinary CAN packets. The next brief introduction was about the message instructions used in the flashing process.

Table2 were the UDS instructions used in the brush writing process (the following instructions are all 16 hexadecimal).

Table 2. Service used during flashing[3]

Service Name	Request	Posresponse	Service definition
DiagnosticSessionControl	10 02	50 02	Enter programming mode
Secret access	27 05 27 06 xx xx	67 05 xx xx 67 06	Unlock ECU for download
Routine Control	31 01 FF 01	71 01 xx	Erase Memory
Request Download	34 xxxxxxxx	74 xx	Request download
Transfer data	36 xx ……	76	Transfer data
RequestTransferExit	37	77	Exit transfer data
Checksum	31 01 FF 02	71 02 xx xx	Check sum result

3. CAN flashing process

3.1 Enter flashing mode

As shown in Figure 3, Software startup mode was composed of Software program startup, initialization stack, global array, etc. The corresponding module code determined to enter into boot or ASW.

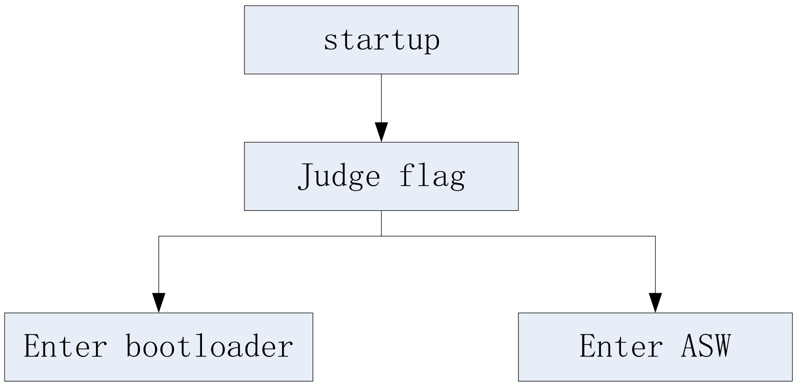

Figure 3. Basic framework of procedure

3.2 Flashing process

After entering bootloader, the whole flashing process contained three parts: communication, downloading, and troubleshooting. The part of communication was based on diagnostic protocol requirements and ensures correct erasure, download, and verification through UDS services.

The flashing process was mainly the operation of flash, including the erasing and downloading of ASW part flash. The part of calibration mainly determined whether the encoded ASW was correct through the diagnostic service. The key goal of the flashing was to erase the data of the program segment and replace it with the data transmitted from the upper computer. The whole strategy of flashing was shown below.

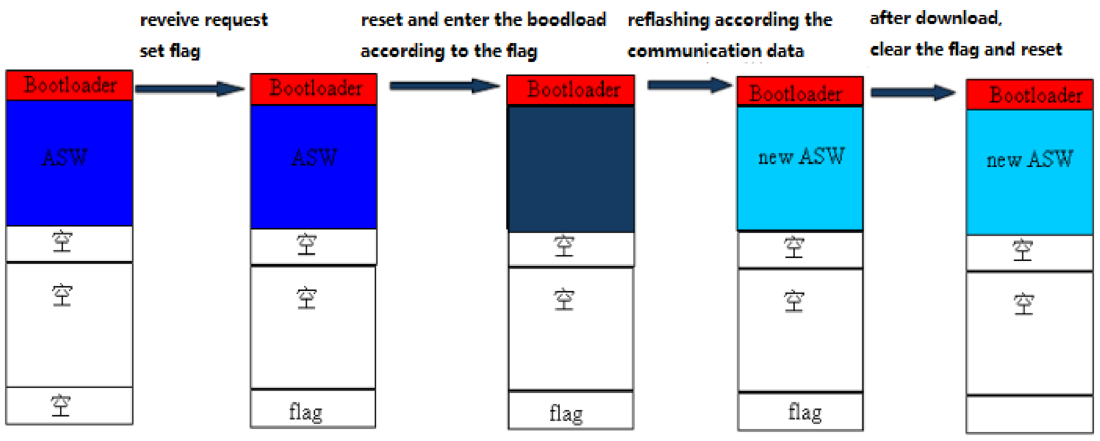

Figure 4. Basic workflow of procedure

As shown in Figure 4, the software program works in the model of ASW. When there is flashing requirement according to the order of UDS on CAN bus, the ASW will set the refresh flag and restart. After the program restarted, it will enter the bootloader programming mode according to the refresh flag. When running bootloader software, program will maintain the communication with the host device, while receiving the data sent by the host device and updating the ASW part.

4. Project development

4.1 Project design

In the project, we developed a bootloader with the example of MC9S12XET256, a 16-bit single-chip microcomputer of Freescale 9S series.

This chip was composed of two Block with 256K flash. Each Block was 128K, divided into 128 sectors averagely. As MC9S12XET256 is a 16-bit microcontroller, the largest address is 64K, therefore, from the chip designing, the concept of page is adopted which is to use a 16K absolute address to map other memory addresses. As shown in Figure 5, the Flash resource allocation for the entire chip is presented.

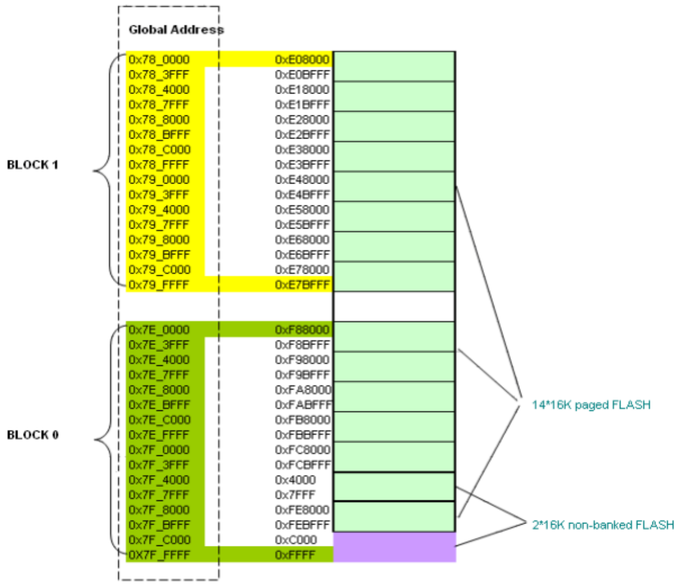

Figure 5. MC9S12XET256 flash address correspondence

MC9S12XET256 has two different erase Flash modes, one is erase command by block and the other is erase command by sector. Block erase command could erase the entire block at one time, i.e. 128K. Sector erase command erase only one sector at a time, i.e. 1K. As shown in table3, the time parameters of block erase command and sector erase command are described.

Table 3. The time parameters of block erase command and sector erase command

XER256 flash erasing time		
flash action	time detect by MCU ticks	time detect by oscilloscope
erase all block(E0-E7)	870*0.128ms =111.36ms	111ms
erase one sector	177*0.128ms =22.656ms	22.8ms
flash module working frequence:181.8KHZ		

In software designing, the bootloader section is only in the last section of Blcok0, that is the address 0xC000-0xFFFF (the purple section in Figure 5). As shown in Table 3, it takes 111ms to erase block1 with block erase commands. However, it would take 22.8ms*16*8=2918.4ms with block erase commands, which is 26 times with block erase commands. Therefore, block erase commands are chosen to erase Block1 flash while sector erase commands are used to erase Block0 flash. According to the above time parameters, the total erasing time was 111 + 22.8 * 15 = 453 ms. In the flash erasing strategy, program uses the background task for erase job, that is, after receiving the erase command, the communication layer of ECU will start an erase task at background. When the erase task finished, the communication layer will send an positive response to the host device.

After the flash erased, the ECU can update the ASW program section with the data sent by the host gradually. The strategy is as followed:

1) Firstly, the ECU receives the request from the host and determines whether the address and data length are valid. If the address and data length meet the expectation, the ECU will inform the host the maximum number of bytes which can send each time according to its own conditions.

2) The host transfers data according to the maximum number of bytes specified by ECU.

3) The host sent all the data in this round and requested the end of this round. Then the ECU checked the data, and judged the validity of the data in the process of communication. If the checksum passed, the next round of transmission will begin.

4) Repeat the strategy above until all data updated.

4.2 Fault handling

In the process of flashing, there will be some abnormal states, such as communication interruption, power failure, flash operation failure, wrong checksum results, etc. When these abnormalities occurred, the program cannot run normally. Therefore, it needs some special methods to deal with those situations.

1) Communication interruption

When communication be interrupted, the ECU cannot receive any data from CAN bus because of external interference, or the error of software and hardware. To solve this problem, we set an approach for special fault handling in the diagnostic layer. The ECU automatically go back to default mode and restart when P3 time out which is 5 seconds. After ECU reboot, the host will quit the operator of refreshing because it cannot receive the response from CAN bus. As the communication devices are re-initialize and restored to its original state after reset. The re-flash processes are to its beginning and can be re-started again.

2) power off

Power failure is always an important problem in fault handling. During the refresh process, the problem about power connection or power supply will lead to the ECU failure. When the power supply is restored and the ECU will restart. Therefore, judgment of the current state is required. Different states entered into different program mode. If the refresh has just begun and the flash has not been erased, the program would enter the ASW segment. When the flash had been erased or was in the state of flashing , the program would enter bootloader segment.

In order to prevent program exceptions from entering into the ASW segment, the power on detection function need be added. That is, after power on, it need be determined whether the last few bytes of each program are the value of exception. If the last few bytes of each program is the value we excepted, indicating that the ASW segment is normal and the process could enter into ASW segment. Conversely, the process must be forced to enter into the bootloader segment.

3) Other faults

In addition to the above several common faults, there were several other faults in the diagnosis of anomalies, such as flash misoperation, flash erasing and writing mistake, incorrect condition about the download requirements, address cross-border, too long data length, etc. These faults are abnormal phenomena caused by the communication process. Since they occurred only after the flash is erased when the program segment was executed in boot, the strategy was to restart and switch the refresh state to the state until downloader sent a refresh request.

5. Experiment and data

After the development of controller and matching the corresponding host device tools [5], the program's flash are completed. In the process of flashing, some occasional problems always occurred. Therefore, it is needed to simulate the robustness of some scene flash platform to be tested [6]. Based on this platform, this paper describes several simulations of various fault scenarios when flashing. The test results are shown in Table 4.

Table 4. Flashing writing function test (part)

Test project	Result
Flash Ecu for 10 times continuously	Pass
Flash Ecu for 10 times with 10 seconds interval	Pass
Communication broken and re-flash	Pass
Injecting error during flashing ,and re-flash	Pass
Reset power during flashing ,and re-flash	Pass
vehicle condition unsatisfied, forcing to flash, whether refused re-flashing	Pass
Wrong checksum , whether quit the re-flashing	Pass
During flashing， modify ECU mode, whether quit the re-flashing	Pass
Wrong password, whether refused re-flashing	Pass

6. Conclusion

This paper described the use of UDS in flash and introduced complete design and completion of the ECU flash through the S12 chip. In the future, There will be consistent growth in ECU flashing , and UDS usage in bootloader will become more and more popular .

References

[1] WANG Chun-hua, ZHANG Yu-wen, HU Ji-kang. Design and Implementation of Fault Diagnosis System for BCM Based on UDS [J]. Auto Electric Parts,2017(08):41-45.

[2] CHEN Zi-lin, SONG Lei-feng, ZHANG Long-gang, DONG Hai. Vehicle Diagnosis and Design Method Based On UDS [J]. Auto Electric Parts,2017(04):14-17.

[3] ISO 14229-1: 2013, Road vehicles - Unified diagnostic services (UDS) --Part 1 : Specification and requirements [S].

[4] ISO 15765-2: 2004, Road vehicles - Diagnostics on Controller Area Networks (CAN) -Part2 : Network layer services [S]

[5] You Changneng . Development of CAN bus UDS diagnostic tool based on Lab VIEW [J]. Electronic Test, 2016 (10): 59-60.

[6] LYYing, SUNYun-xi, LIUDe-li, GUYuan-ye, SUN Yun. ECU Bootloader Refresh Function Test Method [J]. Auto Electric Parts, 2017 (11): 63-66.