Fault Tolerant Protocol for CAN Flash Programming

Ho Gi Jung[1], Jea Young Hwang[1], Pal Joo Yoon[1] and Jai Hie Kim[2]
[1]MANDO Corporation, Korea
[2]Yonsei University, Korea

Abstract

This paper proposes a fault tolerant protocol for CAN (Control Area Network) flash programming. Recently, the need of CAN flash programming is rapidly increasing because the number of installed ECU (Electronic Control Unit) increases noticeably and the adoption rate of CAN is very high. Although some development tool companies and semiconductor companies provide solutions, there is no standards for CAN flash programming until now. Furthermore, there is always a risk, when programming an application into flash memory using downloaded programming algorithms, that an external disturbance may corrupt the programming process. This paper suggests that the standards of CAN flash programming should include mechanism for the recovery from faults and certification process.

Interaction diagram between host PC and ECU summarizes the messages and includes an additional phase for the certification of the CAN communication capability of ECU. The state diagram of ECU with respect to CAN flash programming is developed and includes additional states for the recovery from faults and the certification of CAN communication. Proposed protocol guarantees that ECU can be programmed via CAN without the risk of being dead node. This paper shows the feasibility by summarizing the experimental results.

Keywords: CAN, Flash programming, Fault tolerant, Protocol

1. Introduction

As conventional mechanical components are replaced with electronic implementations, automobile installs more and more ECUs(Electronic Control Unit). Furthermore, almost every ECU adopts CAN(Control Area Network) to share the information of sensors and controllers. According to the popularization of CAN-installed ECU, the needs of calibration and diagnostics using CAN are increasing rapidly. OSEK COM[2] defines the interaction layer for the purpose of diagnostics, which supports continuous data transfer. Established KWP 2000[3] on K-line tends to be replaced with KWP 2000 on CAN, which is based on OSEK COM[4]. CCP(Calibration on Can Protocol) defines the protocol of calibration and diagnostics using CAN communication[5][6].

After the calibration and diagnostics with CAN takes root successfully, it is expected that CAN will improve the maintenance of ECU software. If ECU software can be updated by widely spread CAN, automobile makers can update ECU software ordinarily in cooperation with service network and can avoid expensive recall eventually. Recently, protocols for the software update with CAN are published by several development tool companies[8][9] and semiconductor companies[7][10][11]. However, they are not standardized like the calibration and diagnostics with CAN until now and they are a kind of implementation based on the primitive functions of OSEK COM and CCP.

For the implementation of CAN flash programming, robust treatment of faults during programming process is as important as the establishment of global standards[10]. Because ECU adopting CAN flash programming generally does not have a special hardware for the flash programming and mode selection, ECU can become dead by the failure during programming process if it does not have any solution for the recovery from faults. In spite of the seriousness of the faults during CAN flash programming, published solutions assume that the programming process will not be disturbed and terminates properly [7][8][9][10].

At first, this paper enumerates potential disturbances during CAN flash programming. In spite of various causes and cases of disturbances, malfunctions with respect to CAN flash programming can be summarized into two categories 1) incomplete application, 2) application incapable of CAN communication. Incomplete application can be caused by power down of ECU, communication line disconnection, insane CAN node and shut down of host PC during CAN flash programming. Incapability of CAN communication of ECU after CAN flash programming can be caused by the incorrect setting of CAN driver, the incorrect setting of transaction layer of CCP and the incorrect setting of task.

Suggested CAN flash programming consists of three programs. Each program is developed as a separate project in C language for the maintenance. Sequence diagram between host PC and ECU summarizes the messages and includes an additional phase for the certification of the CAN communication capability of ECU. The state diagram of ECU with respect to CAN flash programming is developed and includes additional states for the recovery from faults and the certification of CAN communication. The state of ECU is coded into a variable located at EEPROM and application can set the state into normal state only by CAN communication. If application could not set the sate into normal state by CAN communication, ECU is set into the CAN downloading state.

Proposed protocol guarantees that ECU can be programmed via CAN without the risk of being dead node. This paper shows the feasibility by summarizing the experimental results.

2. Fault Analysis and Requirements

Table 1 is the fault analysis result of potential disturbances which may occur during programming flash memory via CAN. In spite of various causes and cases of disturbances, malfunctions with respect to CAN flash programming can be summarized into two categories : “incomplete program” and “application incapable of CAN communication”.

Flash programming process consists of four phases : handshaking, erasing, programming and verification. Faults occurring before the erasing phase does not affect the recoverable capability of ECU. Although faults between the erasing phase and the programming phase are various, they are summarized into one category named “incomplete program” because the existing program is destroyed and the new program is incomplete. In the verification phase, process discontinuance itself does not mean the deficiency of recoverable capability ; the discontinuance of verification phase without another defect does not affect the recoverable capability of ECU.

If the CAN driver setting of new program such as CAN ID and communication speed is incorrect, ECU is unable to use CAN communication. If there is a defect in the setting of OSEK COM interaction layer, CCP transaction layer and communication task, data transfer is incomplete and flash programming is impossible in spite of the correct operation of CAN communication. If the new program has wrong setting in the watchdog timer ISR(Interrupt Service Routine) or timer task, ECU will reset repeatedly and will fall into uncontrollable state eventually. Faults in the verification phase excluding recoverable cases are summarized into one category named “application incapable of CAN communication”, because flash programming via CAN is impossible in spite of the success of the erasing phase and the programming phase. In general, the “application incapable of CAN communication” is supposed to occur during development process.

New requirement of CAN flash programming protocol, which reflects the result of fault analysis and refers to the existing implementation, is listed in the below.

- Existing flash programming via K-line uses a specific message during the power-on period or uses a hardware switch as the indicator of booting mode[13]. Because CAN has multiple connected nodes, specific message during the power-on period can not be used. Furthermore, attaching hardware switch to every ECU is too expensive and impractical. Therefore, flash programming must be able to be initiated by CAN message during normal mode without any additional hardware.
- Currently commercialized flash memory generally uses block-wise access control ; While erasing information in a certain block, reading information in the block is impossible. Therefore, the program implementing downloading should reside in RAM area[8][11].
- To acquire the independence of operation context and the convenience of maintenance, it is preferred to develop the CAN flash programming as three independent programs : application program, booting program and downloading program. In this

Table 1. Fault Analysis of CAN Flashing Programming

Phase	Fault	Result	Category
Handshaking	Disturbance before the erasing of flash memory	Abnormal break of programming process	*Recoverable*
Erasing	ECU power down	Previous program is erased partially	**Incomplete program**
	Communication disconnection		
	Insane CAN node		
	PC shut down		
Programming	ECU power down	Programming of new program is not completed	
	Communication disconnection		
	Insane CAN node		
	PC shut down		
Verification	Incorrect setting of CAN driver	Inability of CAN communication	**Application incapable of CAN communication**
	Incorrect setting of the transaction layer of CCP	Inability of data transfer	
	Incorrect setting of the interaction layer of OSEK COM		
	Incorrect setting of communication task		
	Incorrect setting of OSEK OS	Divergence of program	
	ECU power down	Abnormal break of programming process	*Recoverable*
	Communication disconnection		
	Insane CAN node		
	PC shut down		

case, application program includes a specific module for CAN flash programming. Booting program selects which program should be activated between the application program and the downloading program according to the ECU state. Downloading program operates on RAM area and programs ECU flash memory via CAN communication. In the aspect of maintenance, independent C language project for each program is preferable.

- Flash programming process should be recoverable from "incomplete program" fault and "application incapable of CAN communication" fault.

3. Sequence Diagram of Protocol

To meet the new requirements, sequence diagram is developed as the initial step of design procedure. Sequence diagram defines messages between ECU and host PC and shows the interaction between them.

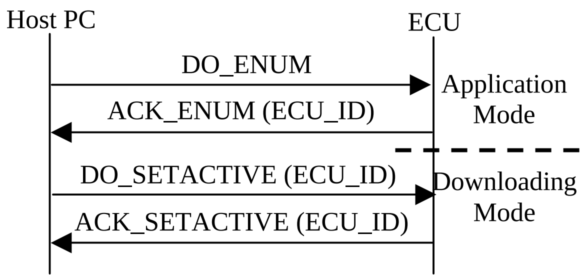

Figure 1. Sequence diagram of handshaking phase

Figure 1 shows the sequence diagram of handshaking phase. Host PC broadcasts a DO_ENUM message on the CAN network using the CAN ID assigned to the host of CAN flash programming. Because of the multicast property of CAN, a node connected to the network can receive every message broadcasted by other nodes[1]. Every node supporting CAN flash programming responses with an ACK_ENUM message. At this time, the priority mechanism according to the CAN ID of each node guarantees lossless data tranfer[1]. Once the node designated as the target of CAN flash programming by the user responses with an ACK_ENUM message, host PC activates the target node by sending a DO_SETACTIVE message. The activated node reports the successful activation with an ACK_SETACTIVE message. Thereafter, the target node reacts to every CAN flash programming related message until it is deactivated or another node is set to a new target node. Inactive nodes do not respond to the CAN flash programming related messages.

Except the first programming case, ECU starts in the application mode in which application program operates. By the DO_SETACTIVE message, the control of ECU is moved from the application program to the downloading program. Because, as mentioned previously, the downloading program is an independent program, the control movement is implemented by the operation context switching using watchdog timer reset or software reset. At this moment, specific variable located in a nonvolatile memory (EEPROM) is checked to record the fact that the existing application becomes invalid. The successful verification phase by the CAN communication between a new application program and the host PC has the exclusive authority of granting the application program to be valid. If the state of application program is invalid, ECU starts with the downloading program, when it is reset, like the first programming case. Therefore, even though CAN flash programming is disturbed by a sudden accident, it is guaranteed that ECU will wake in the downloading mode just by a reset.

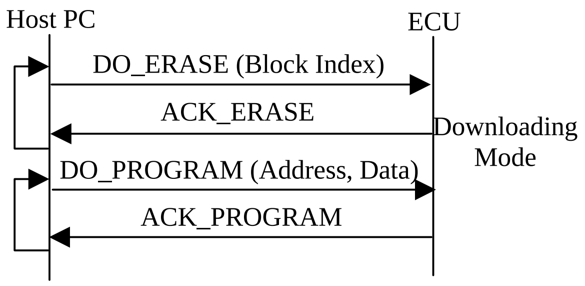

Figure 2. Erasing, programming phase

Figure 2 shows the sequence diagram of erasing and programming phase. The host PC iteratively sends DO_ERASE messages with the block index parameter of flash memory to be erased according to the ECU configuration. The ECU reports that the designated block is erased by sending an ACK_ERASE message. After the erasing phase is finished, the host PC reads a new application program from a hex file and iteratively sends DO_PROGRAM messages with the parameter of address and data. The ECU reports that received data is written in the address by sending an ACK_PROGRAM message. Meanwhile, the data to be programmed can be represented by either CCP CRO(Command Receive Object)[5] format or OSEK COM I-PDU(Interaction layer Protocol Data Unit)[2] format. There are two generally used hex file format : Intel hex format and Motorola S-record format.

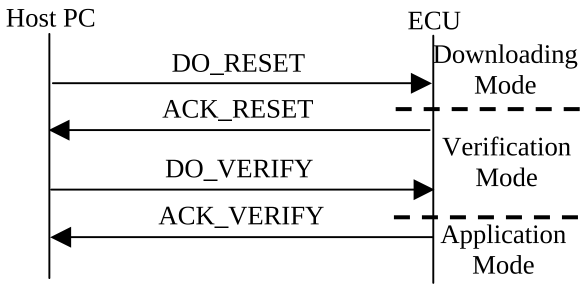

Figure 3. Sequence diagram of verification phase

Figure 3 shows the sequence diagram of verification phase, which is the core of fault tolerance. After the programming phase is completed, the host PC notifies the start of the verification phase by sending a DO_RESET message. The DO_RESET message invokes a control transfer from the downloading program to the downloaded new application program through an operation context switching. Meanwhile, the downloading program sets the ECU state in EEPROM to the verification phase. The new application program acquiring the control of ECU executes the same initialization process as the case of power on reset. After the successful initialization, the downloading specific module of the new application program tries to verify that it has an ability to initialize the ECU and transmit a CAN message without any problem by sending an ACK_RESET message. After the host PC receives the ACK_RESET message, it tests the message reception ability of the new application program by sending a DO_VERIFY message. If the new application program receives the DO_VERIFY message without any problem, all kinds of abilities to fulfill the CAN flash programming are verified. Therefore, the new application program invalidates the verification flag and sets the validation flag of the new application program. At this moment, the new application program resets the ECU and starts a normal operation. A normal application program sends an ACK_VERIFY message during the initialization process to notify that the CAN flash programming is finished.

4. State Diagram of Protocol

Designing the state diagram of ECU state, which is controlled by the CAN messages defined in the sequence diagram, makes the transitions between three programs explicit and concise.

Figure 4 shows the state diagram of proposed protocol. Circle and rounded rectangle represents a state. Double line represents the boundary of a program. Arrow represents a state transition and dotted arrow represents a context switching by direct jump to reset ISR(Interrupt Service Routine). It is assumed that three programs have their own ISRs. Dark circle(•) represents the starting point of the certain level of a hierarchical state diagram. Bull's eye(⦿) represents the ending point of the certain level of a hierarchical state diagram. Developed state diagram has only one ending point at the highest level, which depicts the reset forced by the application program. In this case, hardware reset will occurs as the gray arrow depicts.

DownloadingFlag and ActiveFlag is the variable reflecting the current ECU state and they are located in the EEPROM area. The initial value of EEPROM is treated as TRUE for the DownloadingFlag, but FALSE for the ActiveFlag. Such interpretation of state variable sets the initial state of ECU to Downloading-Inactive state. Erasing operation is implemented by two states : ReadyToErase and Erasing, to guarantee the parallel operation during the execution of erasing command. Such kind of implementation is needed because a general flash memory is asynchronous, in other words, the return from an instruction does not mean the completion of the instruction[12]. Because of the same reason, the programming operation is implemented by two states : ReadyToProgram and Programming. In general, the periodic triggering of finite state machine guarantees the parallel operation by multi-thread, which is an important advantage over an infinite polling loop. Especially, lossless CAN communication needs the implementation suitable for the parallel operation.

Path (A) and (B) in Figure 4 show the state transition sequence of the handshaking phase. After the first flash programming of the booting program using a special hardware such as BDM and JTAG, new application program can be programmed by the designed CAN flash programming. Path (A) is the state transition sequence when there is no valid application program. When the power turns on or hardware reset occurs, the booting program becomes active and switches to the downloading program according to the TRUE value of the DownloadingFlag. Path (B) is the state transition sequence when the CAN flash programming procedure starts from the existing application program. The application program, when receiving the DO_SETACTIVE message, sets the DownloadingFlag to TRUE to represent that the existing application program is invalid and sets the

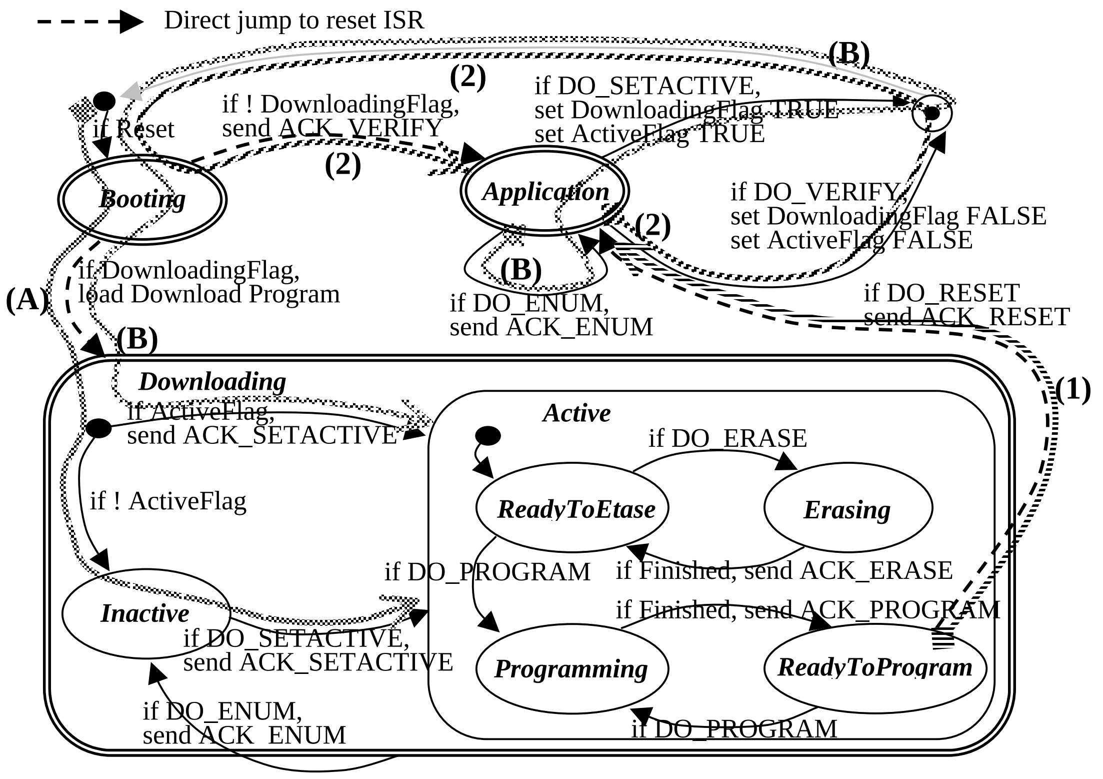

Figure 4. State diagram of protocol

ActiveFlag to TRUE to represent that downloading procedure is onward. Then, the application program resets the ECU to switch to the booting program. After the booting program acquires the control of ECU, it switches to the downloading program according to the state variables. The erasing phase and the programming phase is performed in the Downloading-Active state.

Path (1) and (2) in Figure 4 show the state transition sequence of the verification phase. After the programming of a new application program is finished through the erasing and the programming phase, the downloading program switches to the new application program. Then, the verification phase starts with the transmission of an ACK_RESET message as depicted by the path (1). The new application program verifies its abilities by sending an ACK_RESET message and properly responding to the DO_VERIFY message. After the required capabilities are verified, the new application becomes valid by setting the DownloadingFlag to FALSE and terminates the downloading procedure by setting the ActiveFlag to FALSE as depicted by the path (2).

5. Implementation Issues

To implement the designed protocol successfully, careful consideration is needed for memory architecture, interrupt handling and the realization of state diagram. In general, micro-controller has three kinds of memories : RAM, flash ROM and EEPROM. Although reading method is common, erasing/writing on flash memory or EEROM is different from the direct assignment writing of RAM because it should follow special erasing/writing procedures[12]. Almost every flash memory incorporates more than one protected sections for the downloading program. In addition, general micro-controller uses IVT(Interrupt Vector Table) to handle interrupts.

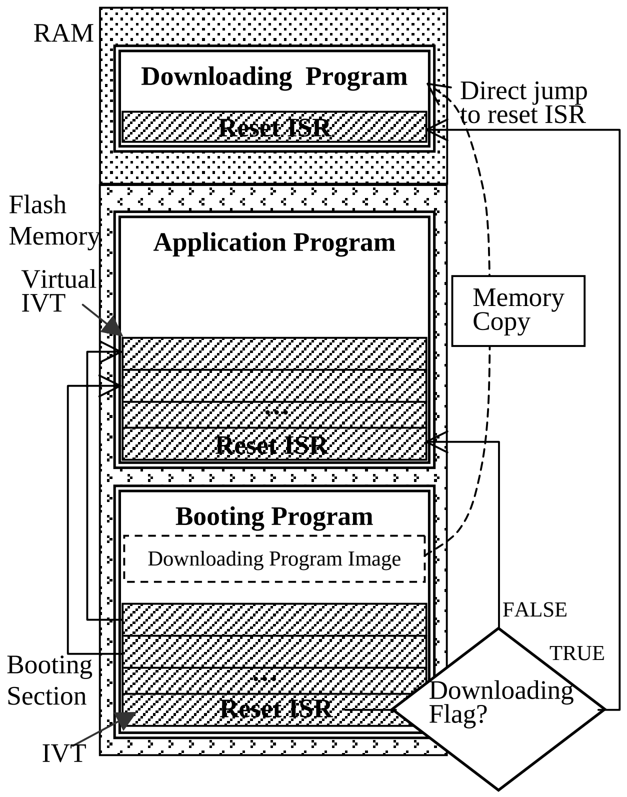

Figure 5. Memory architecture and interrupt handling

Figure 5 shows the location of the three programs on linear address space and describes how interrupts are handled by IVT and virtual IVT. The booting program should be located in the protected section to use the IVT and the application program uses remained flash memory. As mentioned previously, the downloading program operates on the RAM area. The booting program redirects all interrupts to the virtual IVT except reset interrupt in order to locate practical ISRs in the application program area. The virtual IVT is the IVT of the application program and should be located in a designated address to handle the interrupts redirected by the booting program. There are two kinds of mechanisms for the booting program to load the downloading program onto the RAM area : Receiving the program image from the host PC via communication[7][11], Memory copying the program image embedded in the booting program[10][13]. Figure 5 shows the second mechanism.

6. Experimental Result

The developed fault tolerant protocol for CAN flash programming is tested with Motorola MC9S12DP256 used widely in automotive industry. The C compiler of micro-controller is Metrowerks CodeWarrior[14] and PC CAN card is ETAS CANLink compatible with Vector CANcardX.

Figure 6 shows the experimental result that ECU can recover from two kinds of faults, which are selected as the major targets through the fault analysis. Mark (A) in Figure 6 represents the “incomplete program” category. In this case, ECU state variable in EEPROM remains in the Downloading-Active state. Mark (B) in Figure 6 represents the “application incapable of CAN communication” category. In this case, ECU state in EEPROM remains in the Downloading-Active state because the new application does not pass the verification phase. In spite of the different causes, ECU remains in Downloading-Active state. Consequently, ECU can wake into Downloading-Active state just by reset as depicted by the path (1) in Figure 6. Newly started host PC program initiates new downloading process by sending a DO_ENUM message. The ECU switches to Downloading-Inactive state when receiving the DO_ENUM message as depicted by the path (2) in Figure 6. Eventually, ECU is initialized into the same state as the case when the downloading process starts without any application program. It is inspected that all kinds of faults, which occur after setting the DownloadingFlag and the ActiveFlag to TRUE, can be recovered by the same mechanism through diverse experiments. Furthermore, It is inspected that all kinds of faults, which occur in the handshaking phase, can be recovered just by reset because the application program is not destroyed. Finally, it is confirmed that the proposed protocol can recover all kinds of faults occurring at any phase of CAN flash programming.

The advantage of the proposed protocol can be examined by the comparison with the existing protocol which does not consider the possible disturbances. If fault tolerance could not be secured, any trivial mistake after the erasing phase, such as the disconnection of CAN plug and unintentional power off, can cause the

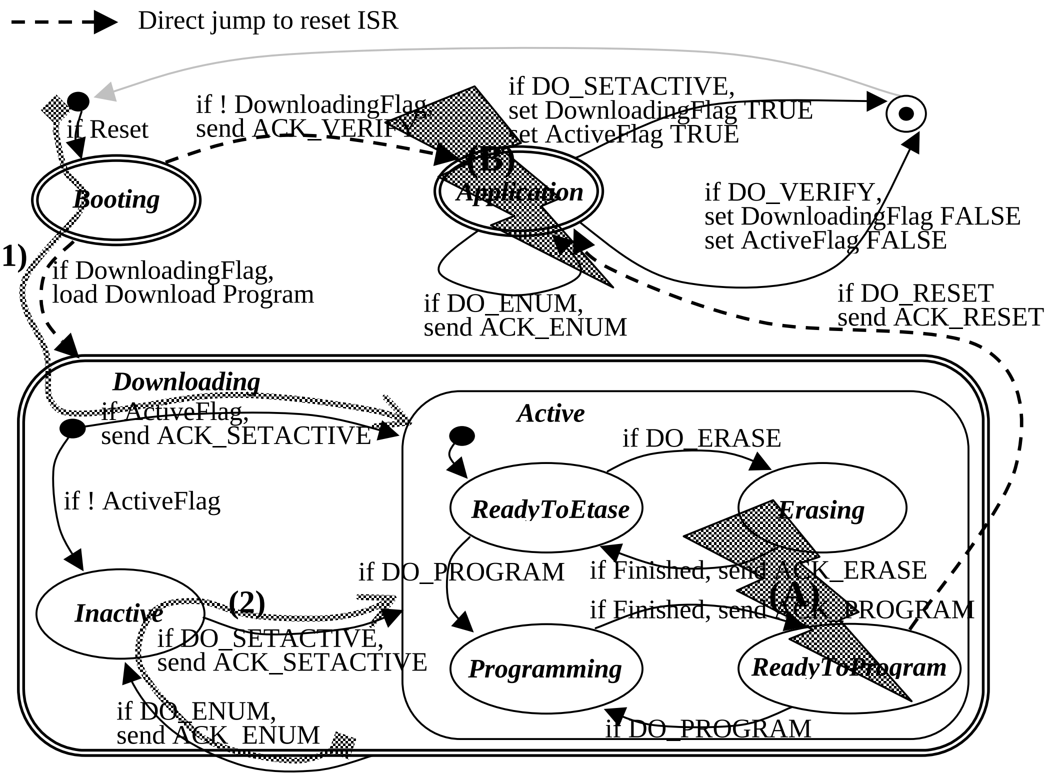

Figure 6. Recovery from faults

ECU to become a dead node. Such a dead node without downloading specific hardware, which is impractical as mentioned previously, can be recovered only by opening the ECU package. It is unacceptable because opening the ECU package means the abandonment of the ECU. At this moment, it must be reminded that one of the important objectives of CAN flash programming is the ordinary maintenance of EUC software through wide service network. Therefore, to make CAN flash programming practical, disturbances able to occur in loosely controlled casual working environment such as a normal garage should be considered.

7. Conclusion

The proposed fault tolerant protocol for CAN flash programming guarantees that ECU can recover from diverse faults by recording the ECU state in EEPROM, which is usual in almost every micro-controller. This paper analyzes possible faults systemically through the fault analysis method. Then a novel CAN flash programming is designed by sequence diagram and state diagram. Additional verification process is developed and the successful communication between host PC and ECU confirms that the new application program is complete and valid. Application program's library, booting program and downloading program are developed as an independent C language project. Furthermore, it is ensured that the protocol can be implemented with small source code and small memory consumption. This paper insists that fault tolerance such as the proposed protocol should be considered in developing CAN flash programming standards.

References

[1] "CAN 2.0B specification", Robert Bosch GmbH, 1991

[2] "OSEK/VDX communication specification version 3.0.1", www.osek-vdx.org, 2003

[3] "Road vehicles-diagnostic system-keyword protocol 2000", ISO14230

[4] Jim Samuel, "Developing diagnostics on KWP2000 and CAN", SAE paper No. 981112, 1998

[5] Kim Lemon, Tammy Dmuchowski and Bruce Emaus, "Introduction to CAN calibration protocol", SAE paper No. 2000-01-0389, 2000

[6] Frank Voorburg, "Rapid application development for embedded systems using CAN calibration protocol", SAE paper No. 2002-01-1170, 2002

[7] "XC166 flash-on-the-fly : A concept to flash via CAN", Application note AP16048 V1.1, www.infineon.com, 2004

[8] Sven Deckardt, "Flash kernel programming on an HC12 microcontroller", Application note AN-IMC-1-002 V1.0, www-vector-informatik.de, 2003

[9] Julien Mothre, "CCP-flashing technical documentation V1.1",ETAS SAS, Rungis, 2002

[10] Ross McKuckie, East Kilbride, "Flash programming via CAN", Application note AN1828/D rev. 1, Motorola Inc., 2002

[11] Martyn Gallop and Joanne McNamee, "HCS12 load RAM and execute bootloader user guide", Application note AN2546 Rev. 0, Motorola Inc., 2003

[12] Stuart Robb, "Fast NVM programming for the MC9S12DP256", Application note AN204/D Rev. 1, Motorola Inc., 2002

[13] Gordon Doughman, "A serial bootloader for reprogramming the MC9S12DP256 flash memory", Application note AN2153/D, Motorola Inc., 2001

[14] "Motorola HC12 compiler manual V5.0.21", Metrowerks, 2000