

Techniques for Reprogramming the Boot in Automotive Embedded Controllers

Thanthry Sreedhar and S. Remya

Delphi Automotive Systems Pvt Ltd

Copyright © 2008 SAE International

ABSTRACT

Typically, automotive embedded microcontrollers will have flash ROM-resident boot (boot loader) software in order to reprogram the application software and/or the calibration data. However, depending on the engineering design and manufacturing process, there may be a need to reprogram the flash-resident boot software area itself. This can be achieved by exploiting the features that the underlying microcontroller offers, with the aim to minimize “brain dead” vulnerability, which is a major reprogramming concern.

This paper first discusses some of the needs for reprogramming the boot area, while some of the methods one can consider to address them are delineated subsequently. The paper summarizes with a word on the implementation results and comparisons using CAN-based serial communication.

INTRODUCTION

The present generation of vehicles employs embedded microcontrollers over one or many serial communication links (referred to as a “vehicle network”) in order to optimally control various systems on the vehicle. These automotive ECUs commonly have flash ROM-resident boot software (generally referred to as a “boot loader” or “boot code” or simply “boot” here). The flash ROM boot loader handles a number of key tasks for these embedded devices that use non-volatile reprogrammable flash ROM for their program code. These tasks, in general, include startup processing, verifying the presence of an application (and eventually executing it), receiving new application software and/or the calibration data over a communication link and reprogramming it over the existing application and/or calibration [1].

Currently, the suppliers, OEMs, qualified dealerships and the service engineers make use of an off-board diagnostic toolset (referred to as a “tester”) that they

hook on to the vehicle network. This is an easy, secure and economical process to upgrade the software and/or calibration. Such a tool (either developed by OEMs or commercially available such as [3], [4]), implementing the same serial protocol as the boot loader resident on the ECU, communicates with the latter in order to reprogram a new application and/or calibration data over the existing ones [2].

NEED FOR REPROGRAMMING THE BOOT

This section examines the requirements for reprogramming the boot software.

ON-BOARD VS OFF-BOARD

These are two broad methods of reprogramming automotive ECUs. On-board reprogramming is changing the ECU's software while it is installed on the vehicle's network, whereas when the ECU is reprogrammed by isolating it off the vehicle, it is called the off-board method. The OEMs prefer the on-board way unless there is a specific reason for offline reprogramming. Various international standards exist for the on-board method, which define the protocol and communication links for diagnostics and reprogramming [5].

This paper concerns itself with those on-board reprogramming techniques for boot software that implement a CAN-based protocol.

BOOT LOADER LOCATION

The boot loader is intended to be a fixed piece of software, i.e., once designed and developed, it is not expected to undergo changes like the application software or calibration data. Hence it is desired that the boot software be completely independent from the application in order to avoid having to reprogram it every time. It would be catastrophic to let the boot or the application erase the boot area during application reprogramming. Thus the boot loader is typically allotted

a block in flash that is separate and physically decoupled from the application and calibration area. This also provides an opportunity to erase and reprogram the boot itself if needed. Perhaps the minimal coupling between the application and the boot is the reset vector of the microcontroller. Design guidelines require it to be part of the boot loader area because of ease of flow right after a microcontroller reset.

BOOT REPROGRAMMING

The boot loader may need to be reprogrammed independent of the application. Following are some typical needs for the same.

Code fix

This is an obvious need. Though the boot software is supposed to be designed and tested for robustness, if it is found to have bug(s), it is critical to have it fixed.

Boot upgrade

The other case is when a feature or protocol update is to be implemented in the boot. This could be driven by the supplier, the OEMs or the protocol/standards governing body. As a special case, consider that the part numbers to identify the ECU and its application software are embedded in the boot flash area (for convenience of reporting it upon the tester's request). When the controllers are in production, if the application software requires a change/update, then the embedded part number has to change to identify the new software in the production ECUs.

BRAINDEAD AND AVOIDANCE

A reprogrammable controller that is unable to communicate with an external reprogramming tool is said to be "brain dead". The top causes for such a situation are:

1. A corrupted reset vector. The controller cannot locate the boot software itself during a cold start.
2. Corrupt boot software. The controller fails to communicate with the tester tool.
3. Corrupt application software. In the warm reset case, the application first communicates with the tester and transfers control to the boot for reprogramming. In this corruption case, the reprogramming may not be achieved.

The time during the reprogramming process when a controller can become brain dead if either the communication or power is interrupted is termed the "window of brain dead vulnerability". The time of this window varies, based on the reprogramming protocol employed and the design.

Clearly, the reprogramming design should have the correct power-on reset vector, boot and application software, and the reprogramming should ideally be carried out in a controlled environment for power. It is also desirable not to try to erase the boot or to lock the boot block (either in hardware or through software). However, in practice, it is close to impossible to completely avoid a brain dead condition. Thus the design or technique should strive to achieve the following goals:

1. Minimize the vulnerability window
2. Design so that the vulnerability window is always known.

METHODS OF REPROGRAMMING THE BOOT

In this section we describe some methods for reprogramming the on-board boot software.

BOOT REPROGRAMMING DURING DEVELOPMENT

The boot loader is normally put in place, either the first time in a controller or thereafter, by means of any supported debugging link during development of that ECU. This is not the subject of this paper.

BOOT REPROGRAMMING DURING PRODUCTION

During production, generally debugging support on the microcontroller will be disabled in order to ensure security against break-ins. Hence any reprogramming at and after this point has to be by way of communication with the ECU. The same communication link will be used by both the boot and the application to talk to the tester for reprogramming purposes. The reprogramming design is also dependent on the architecture of the underlying microcontroller and the features it offers. Of course, a basic care the designers must take is to minimize the brain dead vulnerability, if it cannot be avoided completely.

In order to erase and reprogram the boot, there must be another piece of software in the ECU, which can communicate with the tester.

SECONDARY BOOT TECHNIQUE

This method is suited for those microcontrollers that have a secondary memory, like SRAM or EEPROM, in addition to the flash ROM where the application and boot reside.

Figure 1 shows the method. Right after the ECU is reset, the tester downloads a secondary boot loader (SBL) into the auxiliary memory via a predefined protocol by communicating with the primary or the old boot (PBL). It is worth noting that the original boot refrains from using