

Advanced Firmware Device Manager for Automotive: A Case Study

Joao H. Silva
Visteon Corp.

Abstract

This article describes a case study on firmware device management for automotive systems. More specifically it describes the objectives for the project, the methods used, results and conclusions.

Objectives: The objective is to create an ecosystem to ensure updates to firmware are fast, reliable, and secure and fault tolerant. To achieve this goal, the most advanced technologies in telematics were combined to produce an automotive solution, including: (1) Bootloader; (2) Delta File; (3) File Compression; (4) Encryption; (5) Re-Flash; (6) Bluetooth® wireless technology; (7) USB; and (8) Flash File Systems technologies.

Methods: A pilot project was developed from a case study to understand the complexity of the firmware update problem as it relates to the automotive industry. More specifically, to comprehend the bottlenecks we needed to overcome and to implement the best possible solution. Design and speed performance of the critical path were evaluated. Experiments also were conducted to determine where the software spent time. System components were re-designed where the software spent most of its time.

Results: This paper describes the concepts behind this study from OEM and supplier points of view. It explains the constraints and optimization issues necessary to overcome. In addition, the paper details the critical items that impact overall speed performance of the final design. Finally, this paper shares sequence diagrams, memory maps, workflow diagrams and tables with performance data developed during design.

Conclusions: By the end of this project, it became obvious the traditional Bootloader implementations require major re-designs to improve speed and security. This project also taught how to (1) use the above mentioned technologies wisely, (2) secure the software from attacks, and (3) improve product reliability. It also provided new insight on when and how to use encryption and compression technologies correctly to achieve best possible performance and increase security, as well as how to bind firmware to better flash for improved software update performance.

Keywords: Bootloader, wireless, reliability, fault-tolerance, re-flash, firmware, telematics, Bluetooth, security, encryption, power fail-safe recovery

CITATION: Silva, J., "Advanced Firmware Device Manager for Automotive: A Case Study," *SAE Int. J. Passeng. Cars - Electron. Electr. Syst.* 5(1):2012, doi:10.4271/2012-01-0013.

Introduction

Automotive system designers face significant challenges as automotive ECUs (Electronic Control Units) become more complex. The ability to remotely service and manage the firmware of these ECUs is becoming a critical necessity for the automotive industry. Finding an architecture that supports growing storage volumes without sacrificing performance and providing the intelligence, manageability, and future support for firmware upgrades of varied file sizes and types is a challenge. More specifically, Figure 1, details the most pertinent areas of expertise architects need to think through while designing a Firmware Device Management Systems[4,

5, 6, 7, 8]. Taken together, these primary areas of concern directly impact the architecture and determine the cost of a Device Manager Firmware Update System.

Succeeding sections elaborate on these critical aspects of this technology and describe the experiences how to design and implement this technology in an automotive environment.

Technologies Used

Figure 2 describes the most important technologies automotive system design engineers need to master to design a reliable, easy-to-manage Firmware Update Management



Documents

Document Sharing Hub

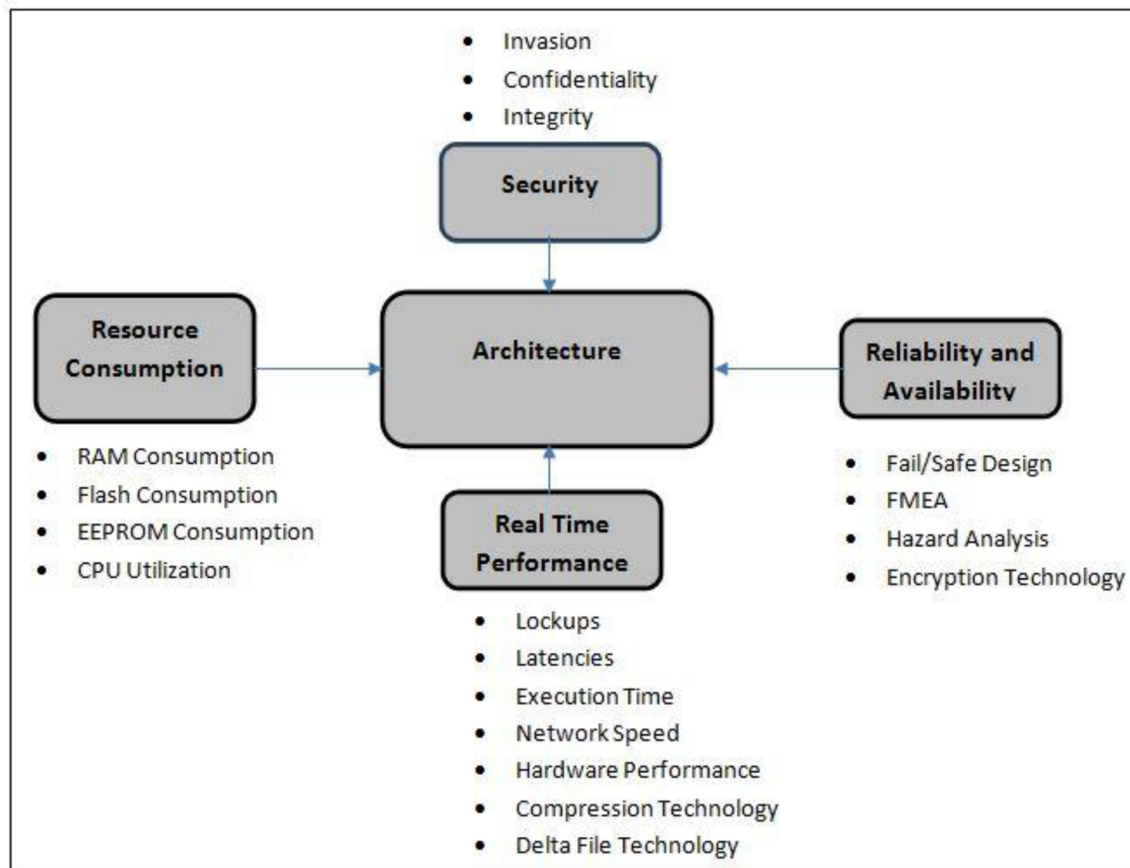


Figure 1. Critical Expertise of a Firmware Update Management System

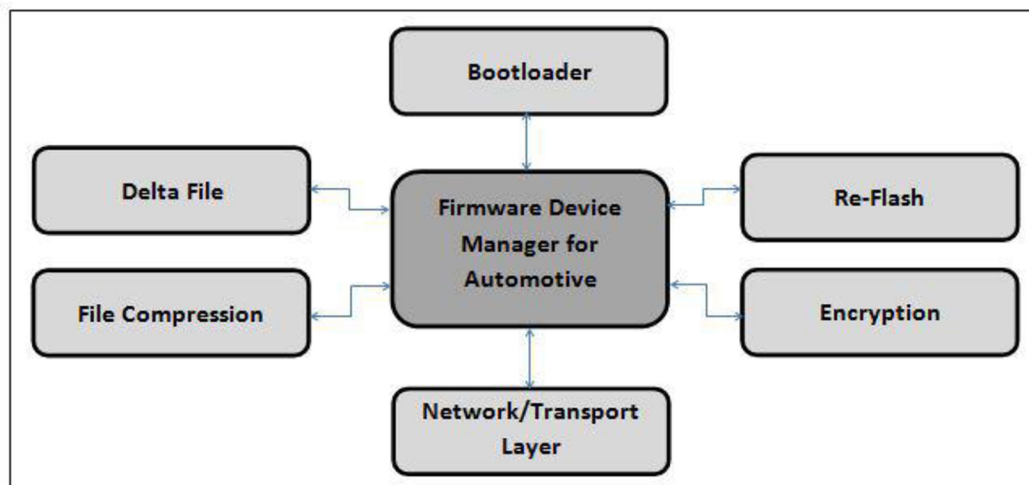


Figure 2. Project Technologies Used

System. The objective is to create an upgrade system that is scalable and capable of accommodating constant growth in the field. Explicitly, the architect uses the technologies depicted in Figure 2 and creates an all-in-one system that integrates a Bootloader, a real-time Operating System and creates an application that can:

- Manage binary images (executable code)
- Perform firmware updates in the field

Notice that some of the technologies in Figure 2 may be not necessary. In fact, some of these technologies can be costly and/or add un-necessary complexity or bottlenecks.

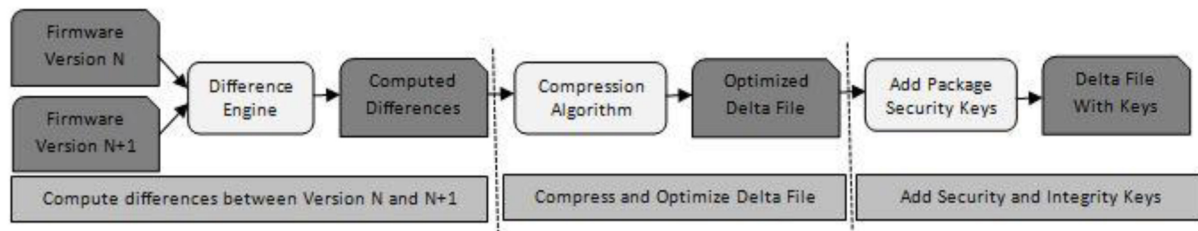


Figure 3. Creating the Firmware Update Package

Firmware Update Systems: Higher performance, lower costs

Ultimately, the major objectives for a project of this type always were:

- To keep the solution simple
- To dramatically reduce the maintenance costs associated with upgrading firmware in the field
- To create scalable solutions and provide maximum flexibility for the end-users
- To improve media access and feature content upgrade within and across vehicle fleets
- To minimize the time it takes to complete a firmware update.

Generating and Deploying a Delta File to Update a Device: Simplest Solution at a glance

Figures 3, 4, and 5 explain the three step process required to complete a firmware update:

- Step 1: Create the Firmware Upgrade Package
- Step 2: Upload the Firmware Upgrade Package into the target
- Step 3: Execute upgrades in the target hardware

Taken together these three steps were designed to establish a reliable, easy-to-manage centralized firmware upgrade solution that can scale to accommodate OEMs and automotive suppliers continuous growth. This solution allows for a clear separation of concerns between OEMs and automotive suppliers. Note: step 1 does not require a vehicle.

Step 1: Creating the Firmware Update Package

During this phase software, engineers prepare the Firmware Update Package (FUP). The generation of a Delta File is the most critical aspect of this phase. Typically, engineers use a difference engine, compute the differences between the two binary file versions, and generate a Delta File.

Compression may (or may not) be needed. It requires a deeper understanding of the hardware performance to determine whether compression may help or not (see Figures 9 and 10). Note: the decision for compression may be dictated by the OEM to optimize the first flash of the module on the manufacturing line using test tools. For the purpose of this paper we are more concerned with the speed performance of firmware updates in the field.

Step 2: Uploading the Firmware Update Package into the Target Hardware

During this phase, refer to Figure 4, we transfer FUP from a PC (or a device) and uploaded in the target hardware.

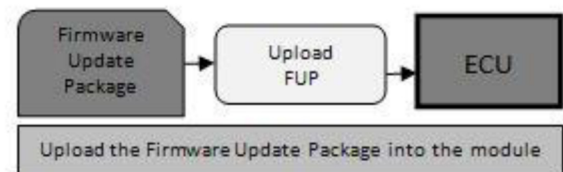


Figure 4. Uploading the Firmware Update Package into the Target Hardware

The following are the most common strategies to upload the Firmware Upgrade Package into the (ECU) Electronic Control Module:

- (1). CD or DVD
- (2). UART and vehicle diagnostics
- (3). Bluetooth® technology when paired with a computer
- (4). DUN (Dial-Up-Network) when paired with a Bluetooth (BT) DUN enabled phone
- (5). USB

Figures 5, 6 and 7 illustrate both Bluetooth technology and USB methods.

Step 3: Executing a Firmware version upgrade in the target hardware

Executing the firmware version upgrade in the target hardware is the last step. Please refer to Figure 8.



Figure 5. Update Using Bluetooth

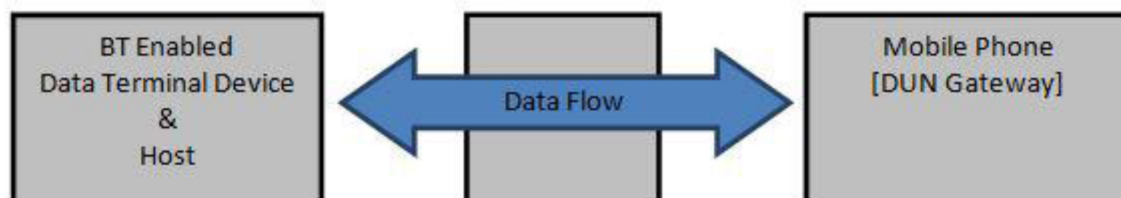


Figure 6. Update Using Bluetooth

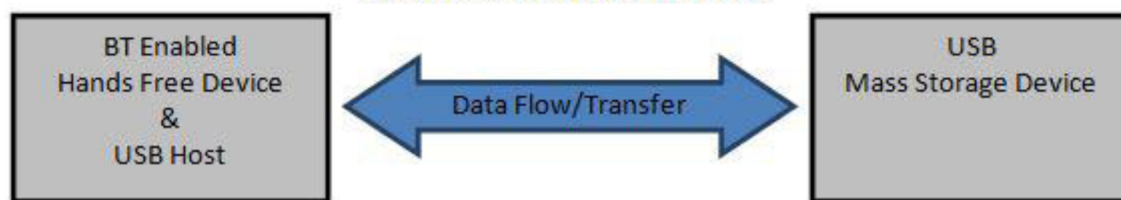


Figure 7. Update Using a USB Mass Storage Device

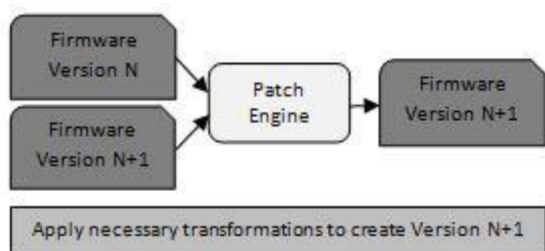


Figure 8. Executing the Firmware version Update on the Target Hardware

The most important aspects of this step are:

1. The total time it takes to perform the firmware update
2. That the firmware update occurs securely and flawlessly
3. When a catastrophic failure occurs, we want to diagnose the problem and always recover

Upcoming sections detail these same aspects.

The Total Performance and Bottlenecks Problem

Bottlenecks take place typically because the automotive module does not have enough of a particular resource and/or the hardware technology available in the ECU:

1. Hardware controller and Clock Speed: Often the micro-controller, which is perfectly suitable for specific automotive design, is inadequate for in-vehicle firmware updates.

2. Network Communication Technology: The network speed (Hardware and/or Software) is another performance concern. The time required to upload the FUP into the ECU is the dominant factor responsible for speed performance degradation.

3. Memory Technology: Both (1) RAM and (2) FLASH technology can become a limiting performance factor contributing for additional speed degradation.

4. Startup time: Time required for initializing the hardware, the RTOS, and readiness to execute the first instruction in the application context.