An In-Vehicle Distributed Technique for Remote Programming of Vehicles' Embedded Software

Radovan Miucic and Syed Masud Mahmud
Electrical and Computer Engineering Department, Wayne State University

Reprinted From: In-Vehicle Networks and Software (SP-1918)
An In-Vehicle Distributed Technique for Remote Programming of Vehicles' Embedded Software

Radovan Miucic and Syed Masud Mahmud
Electrical and Computer Engineering Department, Wayne State University

Copyright © 2005 SAE International

ABSTRACT
From time to time vehicles need to have their software modules updated for various reasons, such as the introduction of new features in vehicles, the need for changing the navigation map, the need for fine tuning various features of the vehicles, and many others. The software in a vehicle's electronic control unit (ECU) can be updated either at a service station or remotely via wireless links. Remote software update has many advantages: it can save consumers valuable time by not requiring them to bring their vehicles to service stations; software in multiple vehicles can be updated in parallel to save auto companies time and money; software in all recall vehicles can be updated in a timely manner, and so on.

There are two main issues related to the remote software update operation. One issue is the bandwidth required for the update operation, and the other issue is the security of the communication links. In another paper we addressed the security issue of the communication links. The cost of bandwidth can be reduced significantly by taking care of multiple vehicles in parallel (multicast process) rather than taking care of one vehicle at a time (unicast process). We explained the multicast update process in a different paper.

Programming an ECU's embedded software requires erasing the ECU's flash memory and then reprogramming the ECU. Erasing the flash memory requires some time, during which the wireless link will remain idle. However, if the wireless link is released while the ECU is erasing its flash memory, then it will take some time to reestablish the link between the vehicle and the remote server. Thus, some bandwidth will be wasted one way or the other. This paper presents an in-vehicle distributed technique to reduce the latency of the remote software update process as well as save the bandwidth of the wireless links.

Every ECU in a vehicle has some RAM buffers to accept blocks of code from an external device before the code is actually written into the ECU's flash memory. If the code size is larger than the total buffer size, then the code is sent to the ECU in several steps. At every step, a part of the entire code is sent to the ECU. Each part of the code is first saved in the RAM buffer, and then it is written to the ECU's flash memory. This process is continued until the entire code is written into the ECU's flash memory. During this process of software update, the link between the external device and the vehicle remains idle for a significant amount of time, which is not acceptable if the external device is a remote unit connected to the vehicle by a wireless link. In this paper, we propose to use the RAM buffers of as many ECUs as we need to keep the entire code that needs to be updated in a particular ECU. This means that first, the code will be distributed among RAM buffers of several ECUs and then the code will be written into the flash memory of a particular ECU. The advantage of this technique is that if the total size of all the buffers of all the ECUs together is larger than the size of the code that needs to be updated, then the link between the external device and the vehicle will not be idle. As a result, the performance of the communication system can be improved. This paper presents a detailed description of the in-vehicle distributed algorithm and compares its performance with that of a non-distributed software update algorithm.

INTRODUCTION
Resource allocation is common among distributed systems. A similar principle can be used in vehicle systems. A vehicle bus and ECUs connected to it are distributed within the system. Nowadays, the speed of the vehicle network and the complexity of its resources are assimilating a distributed system. The workload can be distributed among available ECU resources, so that results can then be used by one or more ECUs in the network. First, we describe a typical reprogramming process. Then, we discuss our proposed solution. At the end, we back up our proposal with a microcontroller and vehicle network survey. In this paper, we present a reprogramming technique using all available RAM resources on the vehicle network.

BOOTLOADER
Background - The bootloader is a part of the embedded software of a typical in-vehicle programmable (IVP) electronic control unit (ECU). The embedded software of the IVP ECU consists of a bootloader (permanent) and operational (changeable) software. The bootloader is the very first code executed on power up or reset. It usually
consists of a reduced serial bus handler, flash erase and write functions, and code verification methods. The main bootloader functionality is to provide the ability for “changeable” software to be reprogrammed. Typical embedded software is presented in Figure 1.

Figure 1 – Execution Flow of Embedded Software in ECU

Description of the current reprogramming process – The programmer, containing the software for the target ECU, is connected to the vehicle bus. The programmer talks to the target ECU. First, the programmer informs the ECU about reprogramming. The ECU prepares for reprogramming by erasing FLASH block(s) of “changeable” memory. FLASH erasing may take a relatively long time. After the ECU has erased FLASH block(s), the ECU informs the programmer that it is now ready to accept data. The programmer sends the first packet. The ECU receives the packet and stores it in the RAM buffer. The ECU responds with a message confirming that the data has been received in the RAM buffer, but the ECU will need some time to write data from the RAM buffer to the FLASH block. The programmer waits for another message from the ECU that will say, “ECU wrote data into FLASH block and it is ready to receive more data”. The programmer sends the second packet, and the previously described steps are repeated until the ECU is completely reprogrammed. A typical reprogramming process that is currently used is shown in Figure 2.

Figure 2 – Current Reprogramming Session
Proposed Implementation - Erasing FLASH blocks and writing packets of data to FLASH memory takes a long time. It takes comparatively less time to send packets on the serial bus and for the ECU to receive packets by storing them in the RAM buffer than the time needed to erase/write the FLASH memory. Available RAM memory of other ECUs on the vehicle bus can be used as "cache" memory. The programmer will distribute the software to the available RAM buffers of the ECUs in the vehicle. After distributing the software to the RAM buffers, the programmer can be disconnected from the vehicle bus.

Detailed Description of the Proposed Bootloader Process Implementation - The programmer initiates the reprogramming session by asking the target ECU (ECU 0) to prepare for reprogramming. ECU 0 starts erasing FLASH memory block(s) of “changeable” memory. While ECU 0 is erasing FLASH block(s), the programmer sends the first packet to the first available ECU (ECU 1) on the bus. ECU 1 stores the received data in its RAM buffer and responds with a message explaining the status of the availability of its RAM buffer. If there is more room available in ECU 1, the programmer will send the second packet of the software to ECU 1. The programmer will continue to fill the ECU 1 RAM buffers until it receives a message from ECU 1 saying that ECU 1 RAM buffers are filled. The programmer will continue to fill available ECU RAM buffers until the entire software for the target ECU has been distributed among the available RAM buffers or until all available buffers are filled. In this paper, we will consider only the case where the entire software for the target ECU is smaller than the size of all available RAM buffers together. The case of the software being larger than the total size of RAM buffers will be considered in a different paper. After the programmer has transmitted the entire software it will then inform ECU 0 or another designated unit of the packets' whereabouts. The designated ECU accepts this packet whereabouts information and then later controls the flow of packets. For now, we will assume that the designated ECU is the target ECU (ECU 0). The process is described in Figure 3.

FEASIBILITY STUDY

An example of one vehicle bus is shown in Figure 4. There are 16 ECUs in the vehicle bus. Let us consider a scenario where all ECUs have the same microcontroller (micro). Let us make several assumptions. The micro has 6 kilobytes of RAM. The size of the “changeable” code is 64 K. The time needed to erase the FLASH block is 1.5 seconds. The time needed to write one byte to FLASH is 10 microseconds. The serial bus speed is 125 kilobits per second. In this simple study we are not considering message overhead, message-processing, latency, and response time and many other things that add to the reprogramming time.
Current reprogramming process – Ignoring the overhead, the time needed to transfer a one-kilobyte packet to the RAM buffer is 65.5 milliseconds. The time needed to write one kilobyte to the FLASH block is 10.24 milliseconds. The total time the programmer has to be on the bus is the sum of the time to erase the FLASH block, the time to transfer data to the RAM buffer and the time to write data to FLASH. The total time is 1.5 (seconds) + 64 x (10.24 + 65.5) milliseconds = 6.347 seconds.

Proposed reprogramming process - Currently when the vehicle is being reprogrammed, none of the ECUs in the vehicle is performing any operations. For example, say a vehicle is at the dealer and the technician is reprogramming the target ECU. Let us assume that 5 kilobytes can be used from each ECU for reprogramming purposes and that 1 kilobyte will be used for storing ECU-critical data.

The total time during which the programmer needs to be on the bus is now only driven by the time during which the programmer transfers software to the RAM buffers. The time for erasing the FLASH block and writing data to FLASH is not a part of the total programmer time. Therefore, the total time the programmer needs to be on the bus is 4.192 seconds. Table 1 shows several microcontrollers and the estimated programmer's time savings in the simplified reprogramming model.
Table I – Programmer's time saved

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>st9f150</td>
<td>6.00</td>
<td>1.5000</td>
<td>125</td>
<td>10.24</td>
<td>6.349664</td>
<td>4.194304</td>
<td>2.15536</td>
</tr>
<tr>
<td>st9f120</td>
<td>4.00</td>
<td>1.5000</td>
<td>125</td>
<td>10.24</td>
<td>6.349664</td>
<td>4.194304</td>
<td>2.15536</td>
</tr>
<tr>
<td>PIC18F6620</td>
<td>3.75</td>
<td>2.0000</td>
<td>125</td>
<td>96.00</td>
<td>12.338304</td>
<td>4.194304</td>
<td>8.144</td>
</tr>
<tr>
<td>PIC18F6620</td>
<td>3.75</td>
<td>2.0000</td>
<td>125</td>
<td>96.00</td>
<td>12.338304</td>
<td>4.194304</td>
<td>8.144</td>
</tr>
<tr>
<td>LPC2119</td>
<td>16.00</td>
<td>0.4000</td>
<td>125</td>
<td>2.00</td>
<td>4.722304</td>
<td>4.194304</td>
<td>0.528</td>
</tr>
<tr>
<td>LPC2129</td>
<td>16.00</td>
<td>0.4000</td>
<td>125</td>
<td>2.00</td>
<td>4.722304</td>
<td>4.194304</td>
<td>0.528</td>
</tr>
<tr>
<td>68HC912D60A</td>
<td>8.00</td>
<td>0.0180</td>
<td>125</td>
<td>1.00</td>
<td>4.276304</td>
<td>4.194304</td>
<td>0.082</td>
</tr>
</tbody>
</table>

CONCLUSION

We developed a new process for in-vehicle reprogramming. The process takes advantage of all ECUs on the vehicle's serial bus. Unused RAM of all ECUs in the programming environment is used as "cache" memory. Using our process, the total time that the programmer needs to be on the vehicle's serial bus is reduced compared to the time using the existing method. Our process can be implemented on top of existing technologies.

REFERENCES

1. "FLASH PICmicro MCUs Break 1 MEGABIT Barrier "
   http://www.futurlec.com/News/Microchip/1MBit.html.
4. "MC912DT128A Data sheet revision 4.0"

CONTACT

Syed Masud Mahmud, Ph.D.
Associate Professor
Department of Electrical and Computer Engineering
Wayne State University, Detroit, MI 48202
Phone: (313) 577-3855
Fax: (313) 577-1101
Email: smahmud@eng.wayne.edu

DEFINITIONS, ACRONYMS, ABBREVIATIONS

ECU – Electronic control unit.
IVP – In vehicle programmable.

CONTACT

Radovan Miucic received B.S. degree in electrical engineering from Wayne State University, Detroit, in 2001 and M.S. degree in computer engineering from Wayne State University, Detroit, in 2002. Currently he is working on his PhD degree, and as a software engineer in automotive industry.

Email: radovanmiucic@hotmail.com