Page navigation

Do You Know the Different OTA Approaches in the Vehicle?

Comparison of OTA Update Strategies

Over-The-Air (OTA) software updates are now an integral part of many consumer electronics products. Apps on smartphones and tablets are supplied with updates practically every day. Applications as well as the firmware of the devices can thus be updated continuously and easily directly at the end user.

In the automotive sector, software updates "Over-The-Air" have already been implemented in some cases, but the functionality is then usually restricted to certain ECUs or parts of the vehicle software. Due to the increasing complexity of vehicle software and its importance for functionality, the need for software updates is growing - even for safety-relevant applications/functions.

Since today's vehicles can contain more than 100 ECUs, an optimal implementation is a real challenge. Vehicle functions distributed over several ECUs must be updated via so-called update campaigns, consisting of update packages for all affected ECUs. This can lead to sometimes complex update scenarios in the vehicle. It is essential that the update processes run automatically, unattended and completely reliably. In the event of an error, it must be ensured at all times that the vehicle can be returned to an operational state, if necessary by completely restoring the previous software version.

Focus on AUTOSAR Classic ECUs

Flash Bootloader as Optimal Addition

This also brings ECUs based on the AUTOSAR Classic basic software into focus, such as door control units from the body domain. These ECUs usually have a so-called Flash Bootloader, which is used to update the application software including the AUTOSAR basic software on the ECU via diagnostics.

Flash Bootloaders have been used for many years to program an ECU software or to update it later in its life cycle. They are comparatively small and yet highly optimized programs that are addressed via diagnostics and erase and rewrite the flash memory. Updates via the Flash Bootloader take place during development, production and in the service shop. At the time of the update, the full bandwidth of the corresponding bus system can be used. In any case, programming takes place in a safe state of the vehicle.

For the use case of "Over-The-Air" software updates, a Flash Bootloader is also an optimal addition (Figure 1).

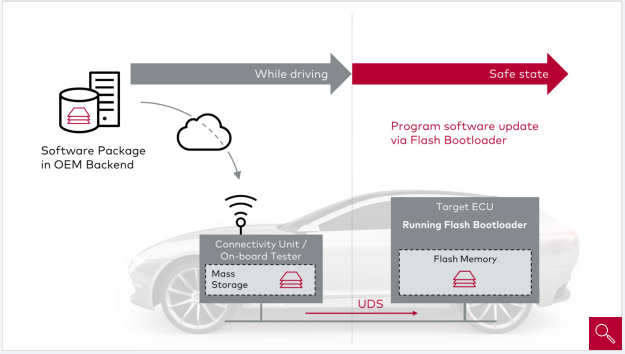


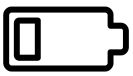
Figure 1: Software update via Flash Bootloader

The new software is transferred wirelessly to the vehicle and temporarily stored on a central ECU, here called a "Connectivity ECU", with sufficiently large memory. As soon as the software is to be uploaded to the target ECU in a safe state, the connectivity ECU starts the update process and loads the software update to the target ECU via a diagnostic sequence - just as the service shop diagnostic tester would do.

Two Limiting Factors in OTA Scenarios

1. During the update process, the vehicle remains in a safe state and cannot be used. This "down time" of the vehicle is usually strictly limited by the OEM for the benefit of customer convenience - this has a considerable influence on the scope or size of the updates.

2. The ECUs involved in the update process must be supplied with power. The remaining capacity of the battery therefore sets a strict limit for the duration of the update.



Houston, We Have NO Problem!

The Way To the Efficient Update Campaign

As already mentioned, there is no alternative to the possibility of restoring a previous software status in the event of a faulty update. Therefore, in extreme cases, a complete reprogramming and rollback of all ECUs involved in the update campaign is required. The above-mentioned limiting factors of downtime and battery capacity restrict the possibilities of a Flash Bootloader in the OTA scenario.

Another possibility is to transfer the data to the respective target ECUs already during normal operation, i.e. while the vehicle is in motion, with storage in a memory area separate from the driving application (Figure 2). The data is not necessarily stored temporarily in the connectivity ECU. Instead, the received data is passed directly to the target ECUs.

This approach has the following advantages:

- The transfer time of the update to the target ECU in the safe vehicle state is being saved.
- Restoring the previous software is possible without further data transmission.

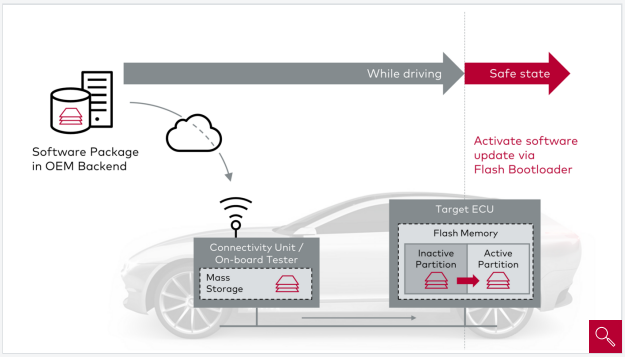


Figure 2: Software download while driving

With concepts that rely on appropriate hardware support for switching between software versions, activation times can be reduced to a minimum. The vehicle therefore remains ready for operation at all times despite the software update.

With Release 19-11, AUTOSAR Classic has published requirements for a Firmware Over-The-Air (FOTA) solution that enables data transfer while the vehicle is still in motion. However, no corresponding basic software module has yet been introduced to the standard.

Vector was an early adopter of FOTA with MICROSAR Classic and has already been offering an extension to the MICROSAR basic software for software download since 2018 that meets the AUTOSAR requirements in particular.

Part 2

Memory Partitioning and Version Switching

A key enabler for a successful OTA update is memory partitioning in the target ECU: The memory must provide a way to cache the software update during normal execution, potentially over multiple drive cycles.

Strategies for Memory Management

This part of the article series therefore focuses on possible strategies for memory management. The various approaches differ mainly on the basis of the required hardware properties as well as the performance of the switchover, i.e. the time that is significantly responsible for the vehicle downtime during software updates.

In order to receive a new software version during the normal execution of the ECU application and to be able to store it temporarily in the ECU, an additional memory is required which can be read and written to independently of the running software. In the following, we assume typical microcontrollers that execute the application directly from flash memory. Therefore, a flash memory, which supports at least one additional partition with Read-While-Write (RWW) property, which is not used for the execution of the application, is required for the application. Such RWW

partitioning allows code to be executed from one partition while writing to another partition. The read partition contains the current software. The written partition is the one into which the new software is written.

This approach provides a solution for storing the software update. However, the new software must also be brought to execution, meaning it must be activated. Overall, the following typical approaches can be considered:

Hardware Supported A/B Swap

A controller with A/B swap capability divides its internal memory into two partitions (also called banks). These two partitions can be assigned a uniform execution address in alternation. It is thus possible that an image linked once can be executed at two different physical positions. Which partition (A or B) is currently active is typically either permanently stored in a hardware register or set by software at each reboot. Thus, a reboot of the controller is sufficient for the switchover. There is practically no downtime of the ECU during the activation phase.

Figure 3 shows an example of the switchover in a system with A/B banks. The physical address range 0xA00000 - 0xC00000 can be read and executed after the switchover via the range 0x00000 - 0x20000.

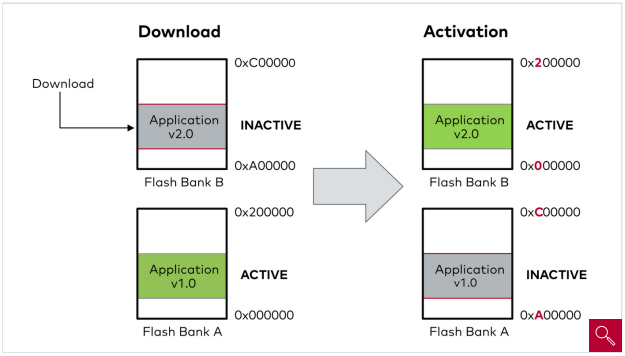


Figure 3: Switchover in a system with A/B banks

Dual Binary Approach and Position Independent Code

Even without hardware support of the address ranges of bank A and B, fast switching can be achieved. But what are the conditions and restrictions for this?

In the dual binary approach, a software version for the application is built both specifically for the addresses for bank A and for bank B. This means that a software update always consists of two different binaries. Only the data that is appropriate for the currently inactive bank is downloaded. So before an update is applied, the correct data is selected based on the active partition. However, this solution has a massive impact on the software logistics of the vehicle manufacturer. This is because he would have to maintain and manage two versions for each software version. Figure 4 shows an example of the scenario in the dual binary approach.

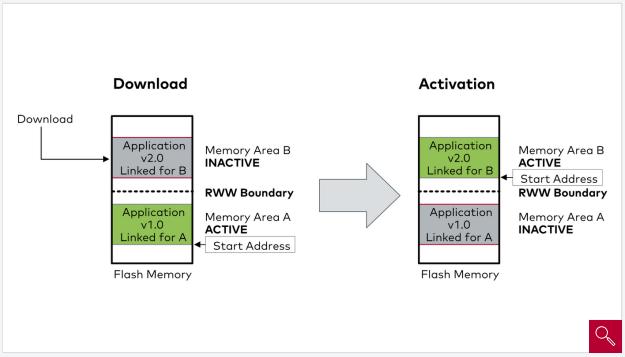


Figure 4: Scenario in the dual binary approach

Another method is to generate the software independently of the actual execution address. This approach is called position-independent code. It is supported by some compilers. In practice, however, it has been shown that support for position-independent code is accompanied by very high demands on the structure of the code. Among other things, this results in disadvantages with regard to execution performance.

Last But not Least: Caching Approach With Backup

If neither the hardware-supported switching of address ranges nor the two alternative approaches mentioned above come into question, there is still the option for a generic way, the caching approach with backup. For this, a cache that is independent of the active partition is all that is necessary. This must of course be large enough to store the software version twice. The idea behind this approach is that the area of the active (i.e. currently running) software remains constant. During the switchover, the area is erased and overwritten with the new software. To meet the requirement for a rollback capability, the current software must be stored as a backup. The buffer will therefore have to be able to store both the new (initially inactive) software and the backup.

This requirement is met as follows

- internal flash with at least two RWW partitions, where partitions not used for execution can store two software states

  or

▸ Availability of additional external memory that can store two software states

Figure 5 shows an example of the caching approach with backup. The creation of the backup can be executed in the background, just like the download of the new software. For the switchover, only the active area must be deleted and written with the software from the inactive partition.

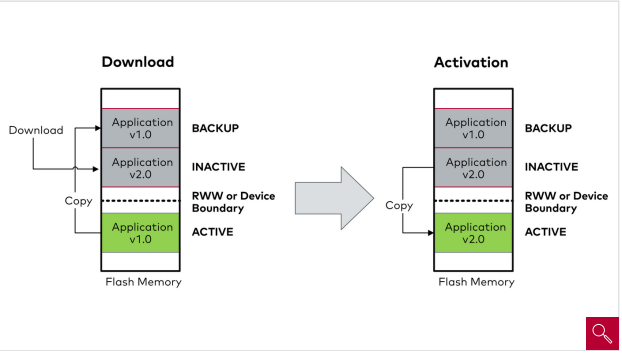


Figure 5: Caching approach with backup

However, the caching approach has a clear disadvantage compared to the other approaches with internal flash: The longer activation time. The advantage, on the other hand, is that it offers more freedom in hardware selection. In addition, unlike the dual-binary approach, the caching approach does not require the management of different binary data for different execution addresses.

The use of external memory should therefore be particularly advantageous for existing ECU projects that are to be expanded with the option of software download. This is a real lifeline when the requirements of the other approaches for internal flash memory cannot be met.

Part 3

Software Download With MICROSAR Classic

This last part of the article series presents the advantages of the Vector solution for software downloads as an extension of the MICROSAR Classic basic software. We will show you how to use it to optimally implement OTA requirements.

Useful Extensions

With MICROSAR Classic, Vector offers basic software according to the established AUTOSAR Classic standard. MICROSAR Classic contains many useful extensions that go beyond the specification to cover the requirements of ECU developers in the best possible way. For example, MICROSAR Classic already contains a solution for software download for several years. This makes it possible to receive and efficiently process software updates during the regular operation of the application.

AUTOSAR Classic has published requirements for Firmware Over-The-Air (FOTA) for the first time with Release 19-11, following the concept of downloading during normal execution. However, AUTOSAR has not yet defined its own FOTA basic software module with interface (API). Nevertheless, the requirements specified in AUTOSAR are already completely fulfilled with the Vector solution, especially in interaction with a corresponding diagnostic application.

Figure 6 shows how the software download cluster integrates into the MICROSAR Classic stack and which interfaces are available.

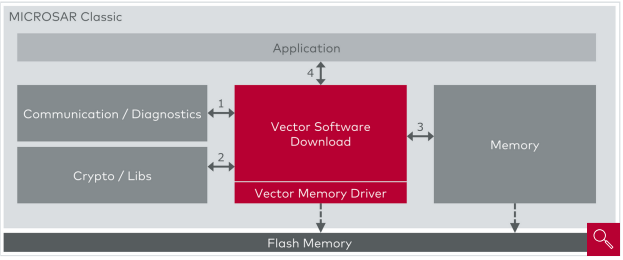


Figure 6: Dependencies to other AUTOSAR clusters

(1) The software download stack provides interfaces to communication and diagnostic modules for data transfer and update control. Usually, these functions are used by the central ECU.

(2) Crypto modules are used for verification (for example, signature check) and processing (decryption) of the received data. Other libraries, for example for checking a CRC over the update data, can also be connected.

(3) A key requirement for a software download in the background is interruptibility, so that the download can possibly be continued over several drive cycles and finally be completed. For this purpose, it is necessary to save the state of the download periodically or when the ECU is shut down. For this purpose, the functionality of the memory modules is used to store data non-volatile in the data flash of the ECU.
Another reason for interdependence is the need to coordinate flash accesses when the underlying flash memory for both data and program flash does not have fully independent access mechanisms. Both the memory stack (for data flash) and the software download stack (for program flash) are users of the shared resource of flash memory. Unless independent access is supported in hardware, concurrent accesses must be resolved by software.

(4) To be able to support project-specific conditions for storage and activation of software updates, queries into the (customer) application are provided. For example, preconditions for

update activation can be implemented in the application by querying CAN signals. Another scenario is the execution of supplier-specific dependency checks to ensure the consistency of the entire software in the ECU.

Stay tuned, because now we'll go into more detail on connections (1), (2) and (3), looking at the individual modules of the Vector software download stack.

(1) OEM-Specific Protocol Implementation

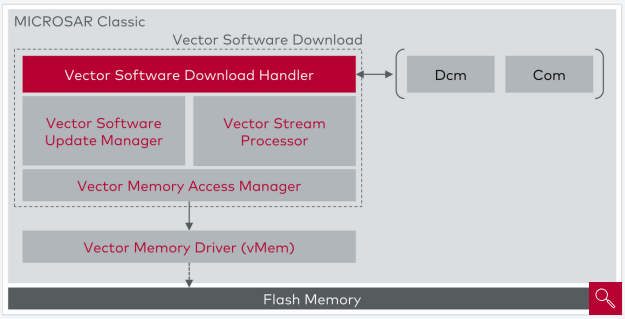


Figure 7: Download handler as a link to central update management

The Vector software download handler implements the connection to the central update control and can receive and process diagnostic messages or messages of proprietary protocols, depending on the requirements. Since protocols and sequences are usually OEM-specific, the download handler at Vector is also implemented in an OEM-specific way. The project-specific queries mentioned above are thereby outsourced to a customer application.

(2) Update Management and Data Processing

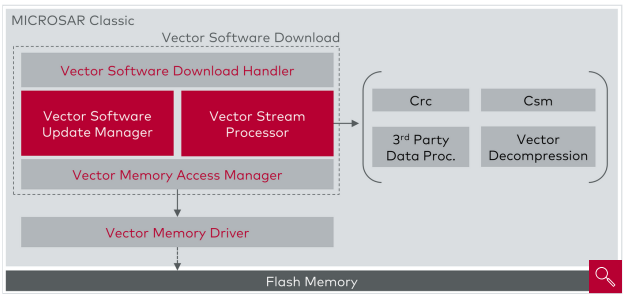


Figure 8: Update manager and stream processor as key components

The two modules Vector Software Update Manager and Vector Stream Processor are the core of the software download stack. The Software Update Manager provides a logical view of the storage areas. So there is an active partition, which stores the currently executed software, and an inactive partition, which is responsible for storing the new software. Optionally, there is also a backup partition. Big advantage: The download handler can work with the logical view and therefore does not have to deal with technical aspects of the flash memory.

Data processing in the Vector Stream Processor module is also performed at the logical level. Data can be passed through, modified or checked in different ways. Concrete processing steps can be flexibly configured, even if they are OEM-specific. Nevertheless, the Vector Stream Processor is kept generic: the OEM use cases can be easily configured with Vector's DaVinci Configurator Classic (formerly DaVinci Configurator Pro) tool. The Vector Stream Processor can use various functions such as CRC and Crypto algorithms from MICROSAR Classic. However, it is also possible to connect third-party libraries.

(3) Vector Memory Stack

To enable software download in the background, the following two memory stack requirements shall be considered:

- A software update is usually stored in the program flash of the microcontroller. The hardware-dependent MCAL-Fls drivers of the hardware manufacturers often only implement access to the data flash. An additional driver for the program flash is therefore necessary.
- Simultaneous access to the data flash and software download to the program flash must be coordinated to avoid conflicts. The application's access to the data flash has priority.

Vector has developed a software architecture for an efficient memory stack that fully meets the above requirements. Figure 9 shows the architecture and the communication paths between AUTOSAR memory and Vector software download modules.

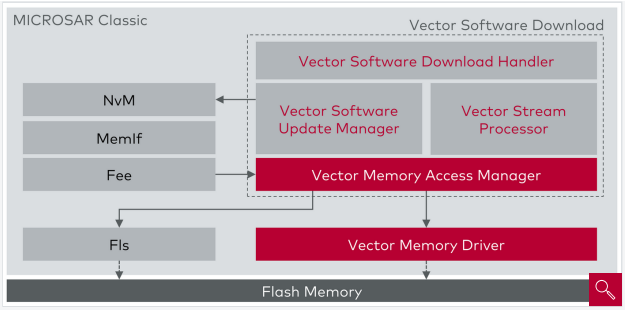


Figure 9: Vector memory stack as an extension of AUTOSAR

The Vector Memory Access Manager is the central module to coordinate any accesses to the flash memory. The call of the AUTOSAR module Fee on AUTOSAR Fls is redirected via the memory access manager. When the Fee module is called, the flash job is first registered in the memory access manager. If there is a free flash resource, the job is forwarded to the Fls driver.

The Vector Memory Access Manager not only deals efficiently with an AUTOSAR Fls driver, but also with a Vector Memory Driver - a module for the general memory driver with an interface newly defined by Vector. The Vector memory driver is hardware-dependent and implements access to the program flash of the microcontroller.

The Vector Memory Access Manager also performs another task: It converts the logical memory addresses to physical addresses. This means that the software update manager remains on the logical level. Only in the memory access manager is the conversion of the logical addresses to physical addresses and the forwarding to the responsible memory driver performed.

The functionalities of the modules Memory-Access-Manager and Memory-Driver developed by Vector were adopted as Memory Access (MemAcc) and Memory Driver (Mem) in the AUTOSAR Classic Standard for the first time with Release 21-11. The connection between the memory stack and the software download stack is also used to maintain states of the download for a possible restart of the update (for example, the download process).

Conclusion: Nothing But Hot Air

With MICROSAR Classic, even challenging update scenarios can be mastered efficiently. The proven basic software takes manufacturer specifications into account and also ensures that the update processes run reliably at all times. With the well-established tool DaVinci Configurator Classic, users can also master all configuration tasks with confidence and in a comprehensible manner.

Over-the-air updates are therefore nothing but hot air - but in the best sense of the word.

MICROSAR.OTA - Basic Software

MICROSAR.OTA provides developers with a solution that goes beyond the AUTOSAR Classic standard. It enables the software download to be performed in parallel with the execution of the driving software in the background. In this process, the received data is stored in a separate memory area that can be accessed independently of the application being executed.

▶ Discover Benefits

Related Pages

MICROSAR Classic

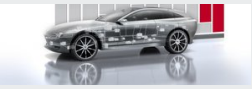
The smart implementation of the AUTOSAR standard.

More information

Flash Bootloader

Compact solution for reprogramming ECUs quickly, efficiently and securely.

More information



DaVinci Configurator Classic

Configuring, validating and generating the basic software (BSW) and the RTE of an AUTOSAR ECU.

Go to Page

AUTOSAR Classic

The production-proven standard for managing automotive ECU complexity.

Go to Page