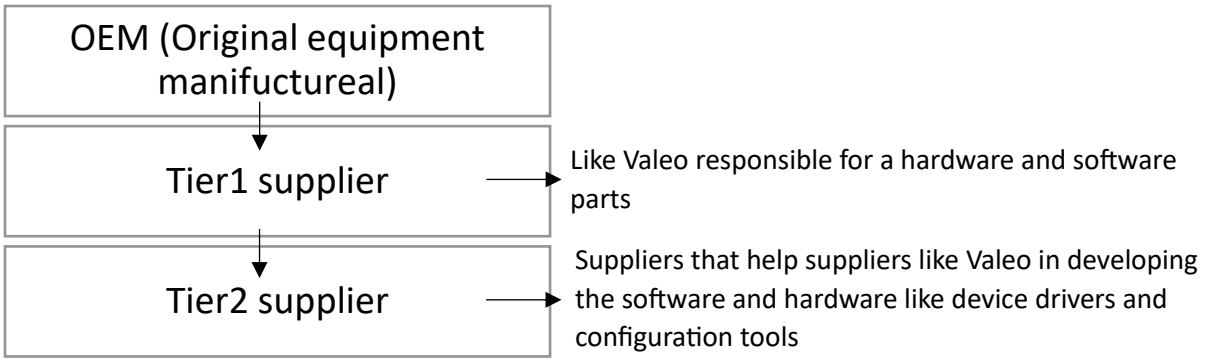
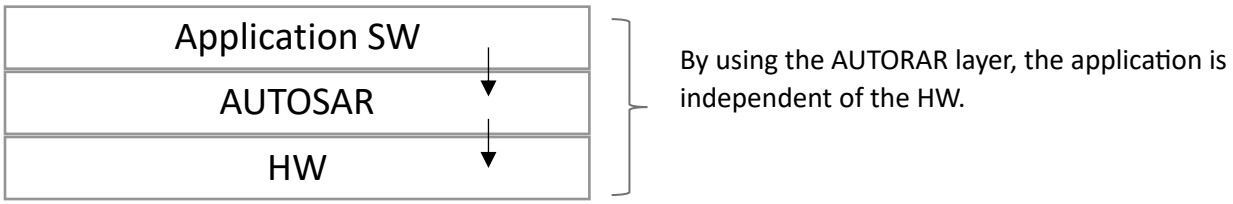
AUTOSAR

(Automotive open system architecture)



- The main idea is integration. Tire1 uses tools from different vendors and these tools are supposed to run together. So that the device drivers for example should have a slandered. For example, we should know what each function does and what parameters does it takes and what return type does it returns.

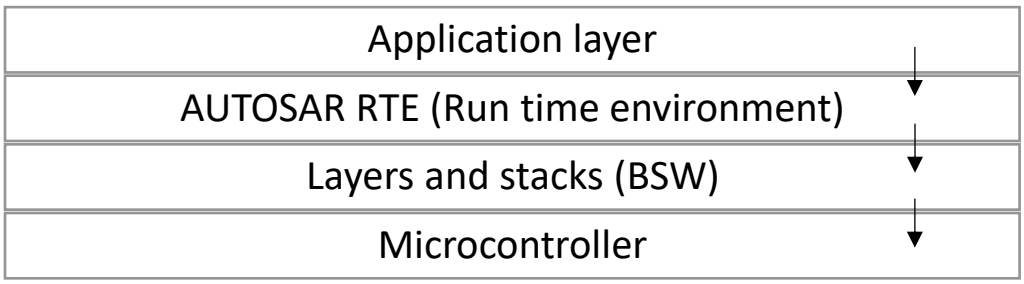
- AUTOSAR is based on <u>layer approach</u>.



- Why should we use AUTOSAR?
    1. The main idea is that the AUTOSAR ***<u>isolate</u>*** the application form the HW and so we can use the same application in more than one project that differs in HW with make some changes.
    2. The ***<u>integration</u>*** point.

- ***Modularity:*** each module is stand alone, i.e., if there is a module that I don't need in my project, I can remove it without affecting any other module.
- Each module consists of:
    1. Static code (.c and .h)
    2. Configuration code.
    3. Make script.
- As a summary, the AUTOSAR help us by isolating the application from the hardware and make the integration process easier.

- AUTOSAR layers

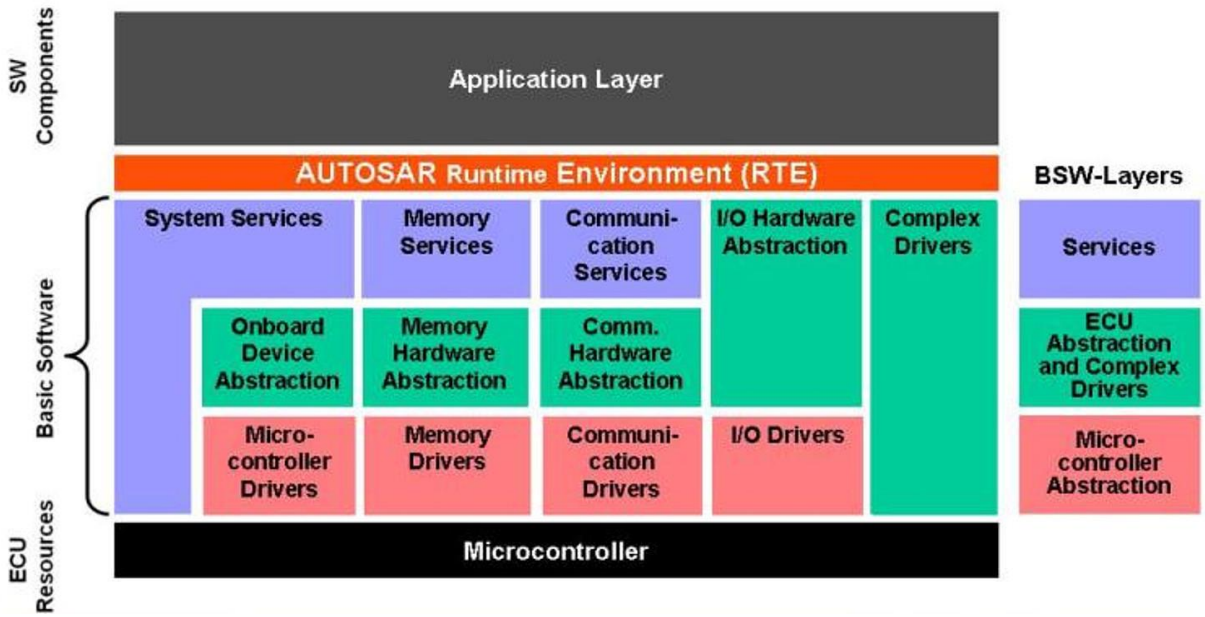


- AUTOSAR has three main topics:
    1. ***Architecture***: here we talk about what layers that our project will be divided into and what are the dependencies between these layers.
    2. ***Methodology***: here we talk about how to make something in our project. For example, what is the steps that we should take to develop a SWC or to make an integration between SWCs.
    3. ***Application interfaces***: here we talk about how SWCs share data between each other.

- In AUTOSAR, we have three main layers:
    1. Application layer
    2. RTE layer
    3. BSW layer
- BSW layer is divided into:
    1. Service layer
    2. ECUAL
    3. MCAL

4. Complex drivers (modules that are not standardized by AUTOSAR)

- BSW layer is divided into ***stack*** and ***function groups***:



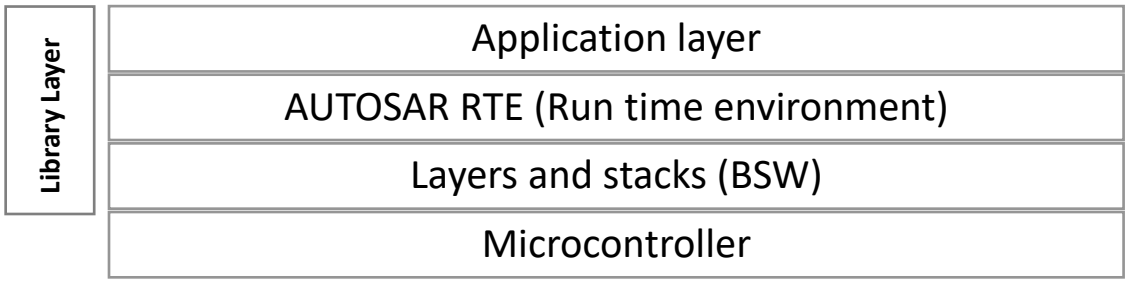
- We can see in the communication stack there is 3 parts, each part is called a ***function group***. And each function group may be in a different layer.
- MCAL layer is HW dependent.

Complex drivers:

- Not specified in AUTOSAR.
- May have very high timing constraints.

Service layer:

- From its name, it provides services to the upper layers like communication or dealing with non-volatile memory. It provides an interface that abstracts us from the lower layers. For example, If I want to receive data form another MCU I will call the function (in the service layer) that receive the data and not care about how the data will be receives (for example, the protocol) and also in dealing with non-volatile memory, we can read and write and not care about what is the type of this memory.



Library Layer:

- Can be called by the application layer and the RTE and BSW.
- Libraries can only call libraries.
- For example, library to calculate CRC.

RTE Layer:

- It is known that the application is a combination of SWCs. These SWCs need to communicate with each other. The RTE layer provides this functionality. i.e., the SWCs of the application layer communicate with each other through the RTE layer. Also, the RTE layer is a bridge between the application layer and BSW layer.

Virtual function bus (VFB)

- Through this concept, the SWCs in the application layer can communicate with each other, even if these SWCs are on different ECUs.
- VFB makes an abstraction between the SWCs so that these SWCs do not need to care about the way(protocol) that the communication in done through.
- VFB concept is done through the ***RTE*** layer.
- To make this abstraction clear we can imagine the scenario of two SWCs on the same ECU communicate with each other through the VFB, we can simply move one of these SWC to another ECU without changing he any of these codes, only the RTE layer code will be changed (it is often a generated code).

AUTOSAR Part-2

- If function A calls function B (which is non reentrant), then function A is non reentrant function also.
- To make a function to be reentrant function
    1. Do not use a global variable (non-constant)
    2. Do not use static global or local variables.

- If there is a function that is a non-reentrant function due to a global resource, we can make it reentrant by using a synchronization mechanism like semaphore.
- If the function uses a global resource and I am sure that this is the only function that uses this resource or this function will not be interpreted, then this function is reentrant.

Sync and Async function

Sync function: can contain a blocking call inside it. And the processor ***cannot*** continue its work during the execution of this function.

Async function: does not have a blocking call inside it. the processor ***can*** continue its work during the execution of this function.

Inline vs normal function

- We use inline function to avoid call overhead.
- The inline function increases memory usage and cannot be used for debugging. If we want to debug, we should use the assembly code.
- In GCC compiler, we must use static keyword in the inline function. Because this compiler supposes that there is another function that has the same name but not inline, so the compiler will decide which function it will use (the inline or the normal). But when there is only one function it must be static.
- Inline keyword suggests (not force) the compiler to make the function inline.

- We can force the compiler to make the function inline.

Callback function

- When a task in a lower layer needs to notify a task in an upper layer, it uses a callback function. Timer ISR for example, we can make this ISR call the callback function.

Scheduled function

- Simple periodic function

Noted about datatypes

- The datatype size depends on the architecture. i.e., it is 8-bit or 32-bit for example.
- In some architectures, it is easier for the CPU to access four bytes of memory than accessing one byte because it will generate less assembly code. Depending on that if there is the counter in a loop, it is recommended to make it 4-bytes counter if it is easier for the CPU as mentioned.

Configuration types

1. ***Pre-compile time configuration***: if there is a change in these files, we will need to re-compile the changed files again. <module name>_chg. It helps us to control some areas of code whether to be taken into the compilation process or not (helps in optimizing the code). And it can be something like a macro that carries the AUTOSAR version for a certain file, we can use this macro to compare it with another macro for another file to check the compatibility for the version and raise a #error if there is an incompatibility issue.

2. ***Link time configuration***: it could be source files and header files. If there is a change in these files, we will need to re-compile the changed files again. <module name>_lcfg.h/c
3. ***Post build time configuration***: configurations that have a certain place in memory. And if there is a change in these files, only the configuration files will be recompiled and reallocated in memory. <module name>_pbcfg.h/c