AUTOSAR

Why AUTOSAR Was Created

- **Standardization:**
  - Before AUTOSAR, automotive companies relied on proprietary software solutions for ECUs, leading to compatibility issues and increased development costs.
  - AUTOSAR provides a standardized software architecture, ensuring interoperability across different manufacturers and suppliers.
- **Modularity and Reusability:**
  - AUTOSAR's architecture allows software components to be developed modularly.
  - These components can be reused across various projects and vehicles, reducing development time and costs.
- **Integration of Advanced Technologies:**
  - AUTOSAR facilitates the integration of complex technologies such as ADAS, autonomous driving, and connectivity within vehicles.
- **Quality and Safety:**
  - By providing a standardized platform, AUTOSAR enhances the reliability and safety of automotive software.
  - It supports compliance with industry standards such as ISO 26262, focusing on functional safety.

Collaborative Synergy: OEMs, Tier 1, and Tier 2 Suppliers

- **OEMs (Original Equipment Manufacturers):**
  - Responsible for designing, engineering, and assembling vehicles.
  - Define vehicle architecture, performance specifications, and features.
  - Manage integration of various components and systems.
- **Tier 1 Suppliers:**
  - Provide complex systems and modules to OEMs.
  - Specialize in specific components like powertrains, chassis systems, and infotainment systems.
  - Work closely with OEMs during vehicle development.
- **Tier 2 Suppliers:**
  - Deliver individual components and subsystems to Tier 1 suppliers.

Introduction to AUTOSAR

- AUTOSAR is a global development partnership of automotive industry stakeholders.
- Standardization initiative by leading automotive OEMs, Tier 1 & 2 suppliers, and semiconductor vendors.
- Founded in 2003 to develop an open and standardized software architecture for automotive ECUs.
- "Cooperate on standards, compete on implementation."

Why AUTOSAR?

- Modularity
- Scalability
- Reusability
- Portability
- Standardized interfaces
- Abstracting from hardware

AUTOSAR Partnership Structure

- **Core Partners:** BMW, Bosch, Continental, Daimler, Ford, General Motors, PSA, Toyota, Volkswagen, Volvo.
- **Premium Partners:** Delphi, Denso, Fiat, Hyundai, Mitsubishi, Nissan, Renault, Siemens.
- **Development Partners:** Infineon, Intel, Magna, NXP.

AUTOSAR Methodology

- AUTOSAR methodology follows a model-based development approach.
- Uses standardized XML-based file formats for software component descriptions, system configurations, and communication descriptions.
- Supports tool interoperability and facilitates collaboration among different stakeholders in the development process.

AUTOSAR Software Architecture

- AUTOSAR architecture is based on a layered software architecture.
- It consists of three main layers: Application Layer, Runtime Environment (RTE), and Basic Software Layer (BSW).

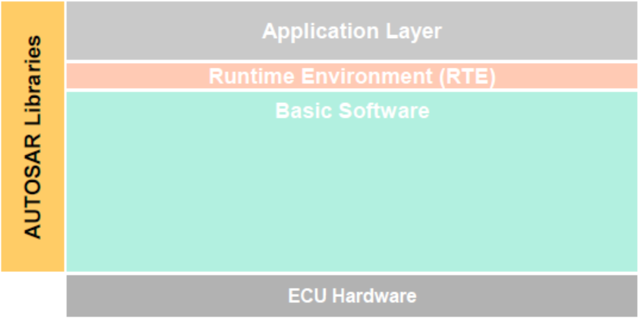

Figure: AUTOSAR Architecture

BSW (Basic Software Layer)

- The Basic Software is divided into four layers: Microcontroller Abstraction Layer (MCAL), ECU Abstraction Layer, Services Layer, and Complex Drivers.
- Provides Services to the Application.
- In charge of running the functional part of software.
  - Ex: Communication (e.g., CAN, LIN, FlexRay), Memory Management, Diagnostics, etc.
  - Contains ECU-specific components.

MCAL (Microcontroller Abstraction Layer)

- Lowest layer of the Basic Software.
- Contains internal drivers, software modules with direct access to the microcontroller and internal peripherals.
- Task: Make higher software layers independent of microcontroller.
- Properties: Hardware dependent, upper layers independent of hardware.
- Drivers mainly provided by microcontroller vendors or by Tier 2 suppliers.

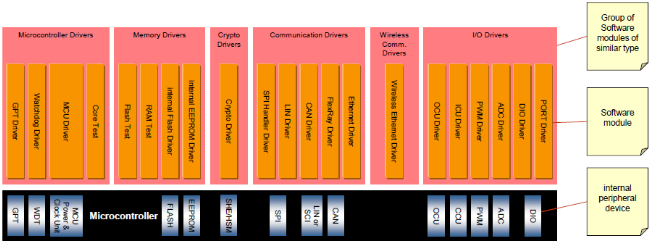

Figure: MCAL

MCAL IO Drivers

- **PORT Driver:**
  - Responsible for configuration of microcontroller's input/output ports.
  - Configuration of parameters: pin direction, pin mode, pin level, change pin direction/mode.
- **DIO Driver:**
  - Responsible for reading and writing digital input/output signals.
- **ADC Driver:**
  - Setup and control analog-to-digital conversion parameters.
  - Obtain the converted digital value of an analog input signal for defined pin channels.
- **PWM Driver:**
  - Initialize and generate pulse-width modulation signals.
  - Control the duty cycle and frequency of the PWM signal.
- **ICU Driver:**
  - Capture the time of a rising or falling edge of an input signal.
  - Provide the time difference between two captured edges.

MCAL Communication Drivers

- **SPI Handler/Driver:**
  - Provides services for communication over the SPI bus.
  - Handles multiple users and multiple devices on the SPI bus.

Example:

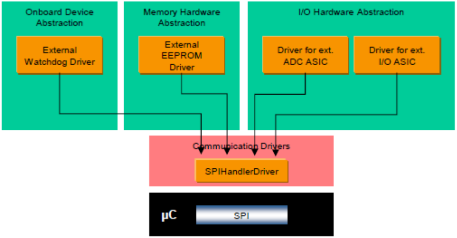

Figure: SPI Handler

- **LIN Driver:**
  - Provides services for communication over the LIN bus.
  - Allows the ECU node to operate as a LIN master.
- **CAN Driver:**
  - Provides services for communication over the CAN bus.
- **FlexRay Driver:**
  - Provides services for communication over the FlexRay bus.

MCAL Memory Drivers

- **Internal EEPROM Driver:**
  - Provides services for reading and writing data to the internal EEPROM memory.
- **Internal Flash Driver:**
  - Provides services for reading and writing data to the internal flash memory.

MCAL MCU Drivers

- **Watchdog Driver:**
  - Provides services for configuring and controlling the watchdog timer.
- **GPT Driver:**
  - Provides services for configuring and controlling the general-purpose timer.
- **MCU Driver:**
  - Provides services for MCU initialization, initialization of the clock system, PLL and clock prescalers, and initialization of RAM sections.

ECU Abstraction Layer

- Interfaces the drivers of the MCAL layer.
- Contains drivers for external peripherals.
- Offers an API for accessing peripherals and devices regardless of location or connection to the MCU.
- Task: Make the upper software layers independent of the ECU hardware.
- Properties: MCU independent, ECU dependent, upper layers are independent of ECU and MCU hardware.

ECU Abstraction Layer Types

- **External Devices:**
  - Contains drivers for external peripherals.
  - Ex: External Flash, External EEPROM, External ADC, External DAC, etc.
- **Interface Module:**
  - Contains functionality to abstract from modules architecturally placed below them.
  - Provides a generic API to access a specific type of device independent of number of existing devices or location.
  - Interface does not change the content of the data.
  - Examples: CAN/LIN/Ethernet Interface, Memory Interface, Watchdog Interface.

ECU Abstraction Layer Modules

- I/O Hardware Abstraction Module
- Communication Hardware Abstraction Module
- Memory Hardware Abstraction Module
- Onboard Device Hardware Abstraction Module

I/O Hardware Abstraction Module

- Group of modules abstracting the location of peripheral I/O devices.
- Task: Represent I/O devices connected to the ECU, hiding ECU hardware details from upper software layers.

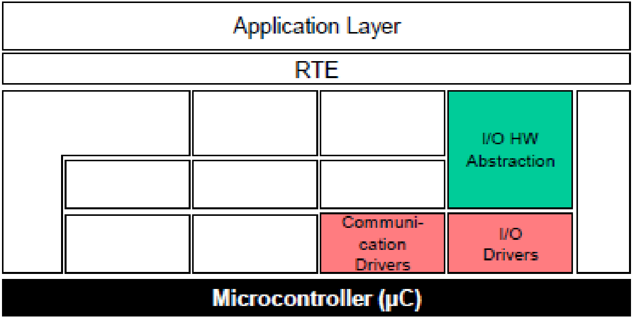

Figure: I/O Hardware Abstraction Module

Communication Hardware Abstraction Module

- Group of modules abstracting the location of communication controllers and the ECU hardware layout.
- Contains communication transceiver drivers like ‘CanTrcv‘, ‘LinTrcv‘, ‘FlexRayTrcv‘.
- Bus interface modules provide a generic API to access a specific type of device independent of its number and location.
- Example: ‘Can Interface‘ module provides access to all CAN channels, whether internal or on-board.

Memory Hardware Abstraction Module

- Group of modules abstracting the location of memory devices.
- Task: Represent memory devices connected to the ECU, hiding ECU hardware details from upper software layers.

Onboard Device Hardware Abstraction Module

- Group of modules abstracting the location of onboard devices.
- Task: Represent onboard devices connected to the ECU, hiding ECU hardware details from upper software layers.

Services Layer

- The highest layer of the Basic Software, providing services to the Application Layer.
- Offers operating system functionality, vehicle network communication, memory management, diagnostics services, ECU state management, supervisor services, and error reporting services.

AUTOSAR OS

- The AUTOSAR OS is a real-time operating system based on the ‘OSEK‘ OS.

Service Layer Modules

- **Det Module:**
  - Collects all development errors reported from BSW modules and takes action based on the type of error and the reporting module.
- **Dem Module:**
  - Production errors are reported to the Dem module, which stores and processes the errors and their associated data.
  - Provides information to the DCM (Diagnostic Communication Manager) module, which communicates it to tester tools connected to the vehicle.

SWS (Software Specification)

- The SWS document describes the software architecture, design, and behavior of an AUTOSAR module.
- Provides detailed information about the module's interfaces, configuration parameters, and functionality.
- Serves as a reference for developers, integrators, and testers.

SWS Structure

- Introduction: Overview of the module and its purpose.
- Scope: Description of the module's scope and intended use.
- References: Lists documents and standards referenced in the SWS.
- Glossary: Defines terms and acronyms used in the document.
- Design Overview: Explains the design principles and architecture.
- Functional Description: Describes functionality and behavior.
- Interfaces: Details interfaces, including ports, signals, and operations.
- Configuration: Specifies configuration parameters and settings.
- Error Handling: Describes how errors and exceptions are handled.
- Testing and Validation: Outlines testing procedures and validation criteria.
- Performance: Provides information on performance characteristics.

Benefits of SWS

- **Clarity:** Provides a clear and detailed description of the module's functionality and behavior.
- **Consistency:** Ensures a common understanding of the module's interfaces and configuration.
- **Reference:** Serves as a reference for developers, testers, and integrators.
- **Compliance:** Helps ensure the module complies with AUTOSAR standards and requirements.
- **Maintenance:** The SWS document can be updated as the module evolves.

Full Software Architecture

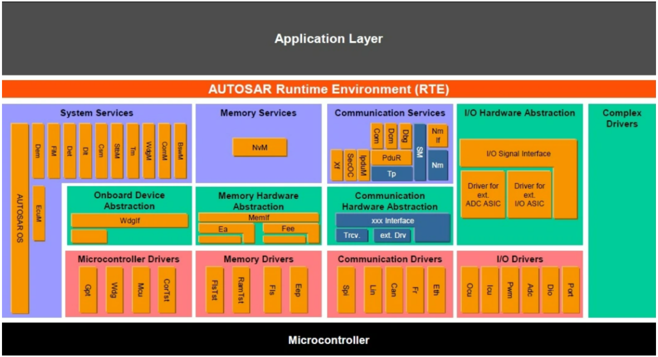

Figure: Full AUTOSAR Software Architecture