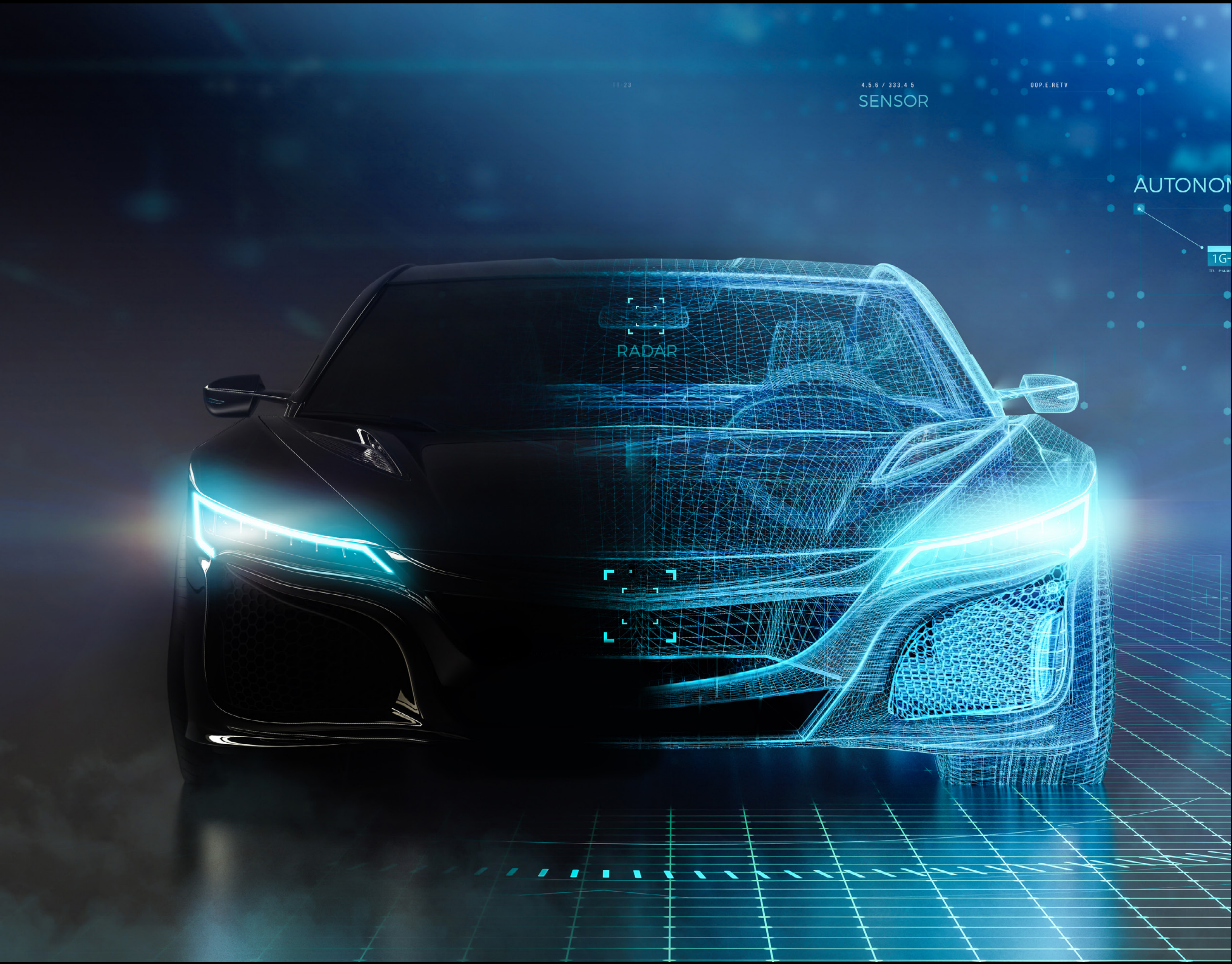
eInfochips
An Arrow Company

ARROW



Understanding AUTOSAR Architecture and AUTOSAR COMSTACK

www.einfochips.com

Software development in the automotive industry differs from other domains/platforms such as Android, Windows, Linux, and web applications. In the automotive industry, software development is dedicated to the Electronic Control Units (ECUs) that meet different standards of security, safety, and MISRA coding guidelines. However, the reusability of this developed software is a big challenge. At present, cars have more than a hundred ECUs and each of them may comprise thousands of lines of codes and functions. All this must be revised from scratch when the hardware (microcontroller, sensors, communication protocol, and interface) is changed because software development is highly platform/hardware dependent. Automotive Open System Architecture (AUTOSAR) standard helps overcome this problem.

AUTOSAR is an open and standardized layered software architecture for the software development of automotive ECUs. It makes an application software independent from the hardware and increases its reusability. It also increases the design flexibility and reduces the development time effectively.

The most important factor in the whole system is communication between various ECUs/subsystems. They transmit and receive data from each other that lets them respond and take decisions based on that data or inputs. In Automotive, various protocols are used for communication between the ECUs such as Controller Area Network (CAN), Automotive Ethernet, FlexRay, and Local Interconnect Network (LIN). AUTOSAR facilitates Communication Stack (ComStack) that provides communication services to an ECU for each protocol.

The most challenging part of ComStack is to understand the configuration for each module and how to link data/info at each layer in different modules. In AUTOSAR, if we want to output a message from the ECU, we need to do the configuration from the application layer to MCAL. At each layer, we need to map/link the data to the next layer. E.g., application layer to RTE (signal mapping) to Service Layer (PDU's mapping) to ECU abstraction layer (PDU to HW object mapping and vice versa) to MCAL.

In this article, we talk about AUTOSAR architecture and understand the AUTOSAR ComStack, irrespective of any protocol (CAN, LIN, and Ethernet), and also discuss the different modules required and their roles in ComStack. We also delve deeper into the configuration of each module from the service layer to MCAL and address the basic configuration for each module. The article gives you a detailed idea of the configuration of CAN ComStack of the modules and the type of configuration they need at different layers and an overview of the multi-protocol stack.

1. Autosar

Considering the increasing complexity of software development in the automotive industry, leading automotive giants (BMW Group, BOSCH, Continental, DAIMLER, Ford, General Motors, PSA Group TOYOTA, and VOLKSWAGEN) formed a group/partnership called AUTOSAR i.e., Automotive Open System Architecture. These all are core partners of AUTOSAR, and the other partner types are premium partners, development partners, and associate partners. Premium and development partners use this standard and cooperate with the core partners and with each other to define AUTOSAR standards. The associate partners use AUTOSAR standards documents that have already been released. It is an associate group formed by the OEM's, Tier 1 automotive suppliers, semiconductor manufacturers, software suppliers, and tool suppliers. It was started in 2003 and they released the first classic platform standard 2.1 in 2006 and later upgraded it to the current release – 4.4.0 (2021).

AUTOSAR is a Layered Automotive Open System Architecture used for the development of software architecture for an ECU. It consists of three layers:

- Application layer
- Run-Time Environment (RTE)
- Basic Software (BSW)

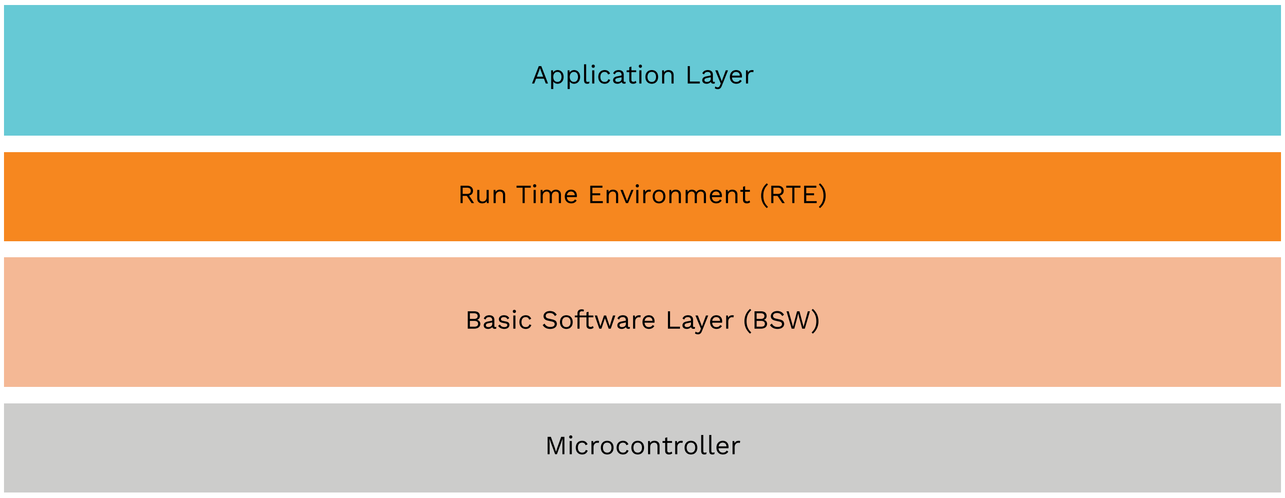


Figure 1: AUTOSAR Layer Architecture

1.1. Application Layer

The application layer is the uppermost layer in AUTOSAR, which contains the application software called the Software Component (SWC). At this layer, the hardware and communication protocol/interface used to exchange data are unknown to the SWC. It is a completely hardware-independent development of application software that only focuses on the final output (E.g., speed/time/any end calculated value) and data.

1.2. Run-Time Environment (RTE)

RTE is the middle layer in AUTOSAR that provides an interface and enables communication between the Basic Software (BSW) and the application layer. RTE separates the application layer from the BSW and enables the Inter ECU and Intra ECU communication. RTE works on Virtual Bus Communication (VFB) that provides the mechanism to connect all these components (SWC, BSW) and enables communication.

1.3. Basic Software (BSW)

BSW is the lowest layer in AUTOSAR that provides a hardware-independent abstraction to the upper layer. It interacts with the actual hardware (microcontroller) and makes the upper layer implementation independent from the hardware.

Further, BSW has three layers:

- System Service Layer
- ECU Abstraction Layer
- Microcontroller Abstraction Layer (MCAL)

1.3.1. System Service Layer

The system service layer is the topmost layer in the BSW that interacts with the RTE and ECU Abstraction layers. It provides communication services, memory services, ECU mode management, state management, and diagnostic services. It also contains an Operating System (OS) for task scheduling and prioritization.

1.3.2. ECU Abstraction Layer

ECU Abstraction Layer is the middle layer in the BSW that interacts with the System Service and MCAL layers. It interfaces and abstracts the drivers of MCAL and makes the higher software layer independent of the ECU hardware. It also contains the drivers for external devices.

1.3.3. Microcontroller Abstraction Layer (MCAL)

MCAL is the last layer in the BSW that is hardware-dependent and is accommodated with the drivers for the hardware (microcontroller). It makes the upper layer independent of the microcontroller. MCAL has direct access to the microcontroller and its peripherals.

2. Communication Stack (ComStack)

In the AUTOSAR architecture, ComStack enables communication between the ECUs and other devices. With the help of ComStack, ECUs can also communicate with the outside world i.e., the external devices (diagnostic tester tool, Smart Sensors, Smart Actuators, and other ECUs). AUTOSAR communication stack supports different protocols like CAN, LIN, FlexRay, and Ethernet.

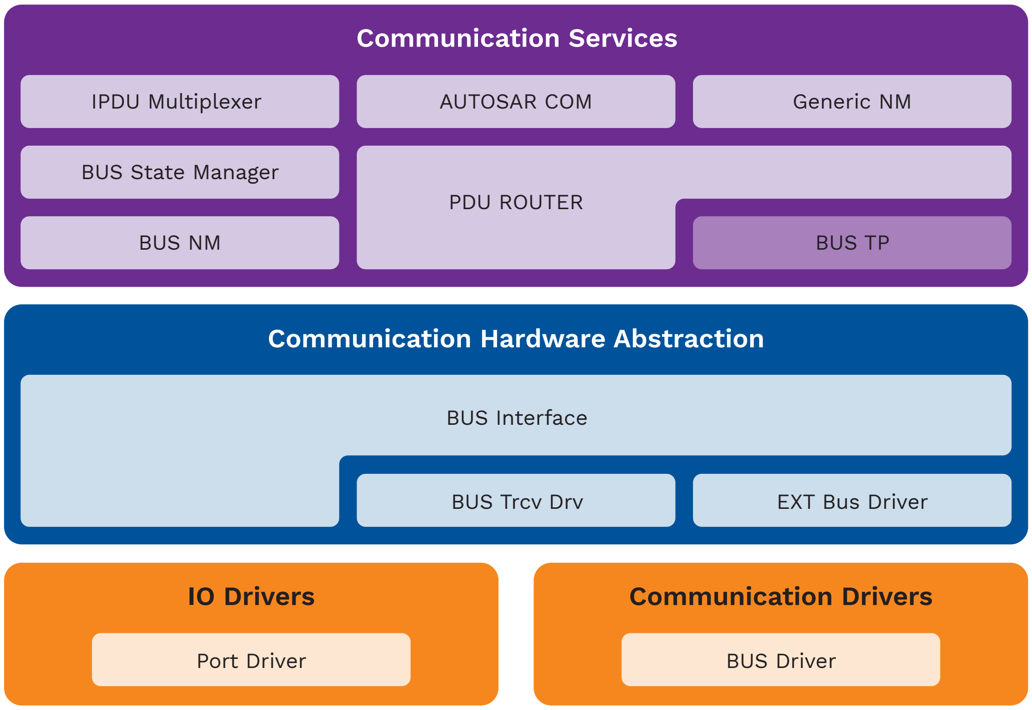


Figure 2: AUTOSAR ComStack

- AUTOSAR provides communication services through ComStack. It's a part of the BSW layer. The functionality of ComStack is divided into three layers. Each layer includes a different module.
- In AUTOSAR, information/data is transmitted and received in terms of signals and PDUs (Protocol Data Units), with the PDU consisting of data and its length.
- Signals are used at the application layer and the SWC exchanges information/data with RTE in terms of signals. Later, those signals map it to a specific PDU in the service layer.
- Each layer has its PDU (refer to sec 5.1). For example, while transmitting from the service layer to the ECU abstraction layer, the service layer adds PCI (Protocol Control Information) into the PDU that helps the abstraction layer to understand it and its contents.

2.1. Service layer

From the ComStack perspective, the service layer provides communication and management services that include the following modules:

- AUTOSAR COM
- PDU Router (PDUR)
- BUSTP
- BUSSM
- BUSNM
- Generic NM
- IpduM

Note: BUS may be CAN, LIN, and Flexray

2.1.1. AUTOSAR COM

- Placed between the RTE and PDUR. It facilitates the signal-oriented data interface for the RTE.
- It maps signals to the PDU during transmission and extracts signals from the PDU during the reception.
- The transmission of the PDUs can be periodic, direct, or mixed.
- Controls the communication transmission like starting and stopping of the I-PDU groups.
- Provides signal-level gatewaying, signal invalidation, and deadline monitoring.
- Provides signal-level access to the application layer and PDU-level access to the low-level modules.
- In this module, the I-PDU (Interaction Layer PDU) is prepared.

2.1.2. PDU Router (PDUR)

- The PDUR manages the routing of the PDUs. Routing is based on the statically defined PDU identifiers and each PDU is configured with its source and destination. There is no dynamic or run-time routing of the PDUs. Up to the PDUR module, PDUs are protocol/bus independent, and later they are routed to a protocol-specific module.
- PDUR can be used for single-cast, multicast, and gateway purposes. In gateway, PUDR receives the PDU from one protocol-specific BUS interface module and routes it to a different or same protocol-specific BUS interface module (CAN to CAN or CAN to LIN).
- The upper and lower modules of PDUR are not fixed. Upper modules are COM, DCM, IpduM, and lower modules are BusTp and Bus Interface. The most common pairs used are DCM with BusTp, AUTOSAR COM with Bus interface, BusTp, or IpduM.
- PUDR receives an I-PDU and transmits it to the next module.

2.1.3. BUSTP

- BUSTP is a transport protocol module. It is protocol-specific (CAN, LIN, FlexRay) that is placed between the PDUR and Bus interface. It provides different services such as data segmentation during transmission and reassembling of data during the reception, flow control, and error detection in the segmentation process.
- It is an optional module that is required when we transmit data, where the length is more than the data length defined in the protocol frame format.
- Most commonly used in diagnostic services or larger data transmission
- It receives an I-PDU from PDUR and prepares an N-PDU (Network Layer PDU) and sends it to the BUS interface. It is used to fragment an I-PDU.

2.1.4. BUS State Manager (BUSSM)

- BUS State Manager is a protocol-specific module. In the vehicle communication network, each network has a unique network handle. BUSSM handles the switching of communication modes of the configured networks depending on the mode request from the AUTOSAR Communication Manager (ComM).
- BUSSM interacts with the ECU Abstraction Layer and System Service Layer.
- It receives an I-PDU from PDUR and prepares an N-PDU (Network Layer PDU) and sends it to the BUS interface. It is used to fragment an I-PDU.

2.1.5. BUS Network Manager (BUSNM)

- BUS Network Manager is a protocol/BUS-specific module that provides an alteration between Network Management Interface (NM) and Bus Interface. It is responsible for handling the transition between the network modes (sleep mode, normal operation, and so on).

2.1.6. Generic NM

- It is a network/BUS-independent module that acts as an adaption layer between the BUSNM and AUTOSAR Communication Manager (ComM).

2.1.7. IPDUM

- It is an optional and protocol/BUS-independent module that is used to multiplex several I-PDUs into one PDU. Multiplexing is done using the PCI.

2.2. ECU Abstraction Layer

From ComStack's perspective, it has a Communication Hardware Abstraction layer that is protocol/BUS-specific and is located between the service layer and MCAL layer modules. It contains the driver for an external device and transceivers such as an external CAN controller, external ADC, CAN transceiver driver, and FlexRay transceiver driver.

- BUS Interface (CANIF, LINIF, etc.)
- Transceivers driver
- External devices driver

2.2.1. BUS Interface

It provides a unique interface to the upper layer module (PDUR, BUSTP) to manage various hardware devices for data transmission and reception such as transceivers and external controllers such as CAN and LIN.

It also provides different services like transmit request services, transmit confirmation services, reception indication services, controller mode control services, and PDU mode control services.

It receives an N-PDU from BUS TP or I-PDU from PDUR and prepares an L-PDU (Data Link Layer PDU). Later, it sends it to the driver (MCAL).

2.3. ECU Abstraction Layer

From the ComStack perspective, it has a communication driver layer that is protocol/BUS-specific and directly accesses the hardware to offer the hardware-independent API to an upper layer.

Communication drivers:

- Ethernet Driver (ETH Driver)
- Controller Area Network Driver (CAN Driver)
- FlexRay Driver (FLX Driver)
- Local Interconnected Network Driver (LIN Driver)

2. CAN Communication Stack (CAN ComStack)

In the AUTOSAR architecture, ComStack enables communication between the ECUs and other devices. With the help of ComStack, ECUs can also communicate with the outside world i.e., the external devices (diagnostic tester tool, Smart Sensors, Smart Actuators, and other ECUs). AUTOSAR communication stack supports different protocols like CAN, LIN, FlexRay, and Ethernet.

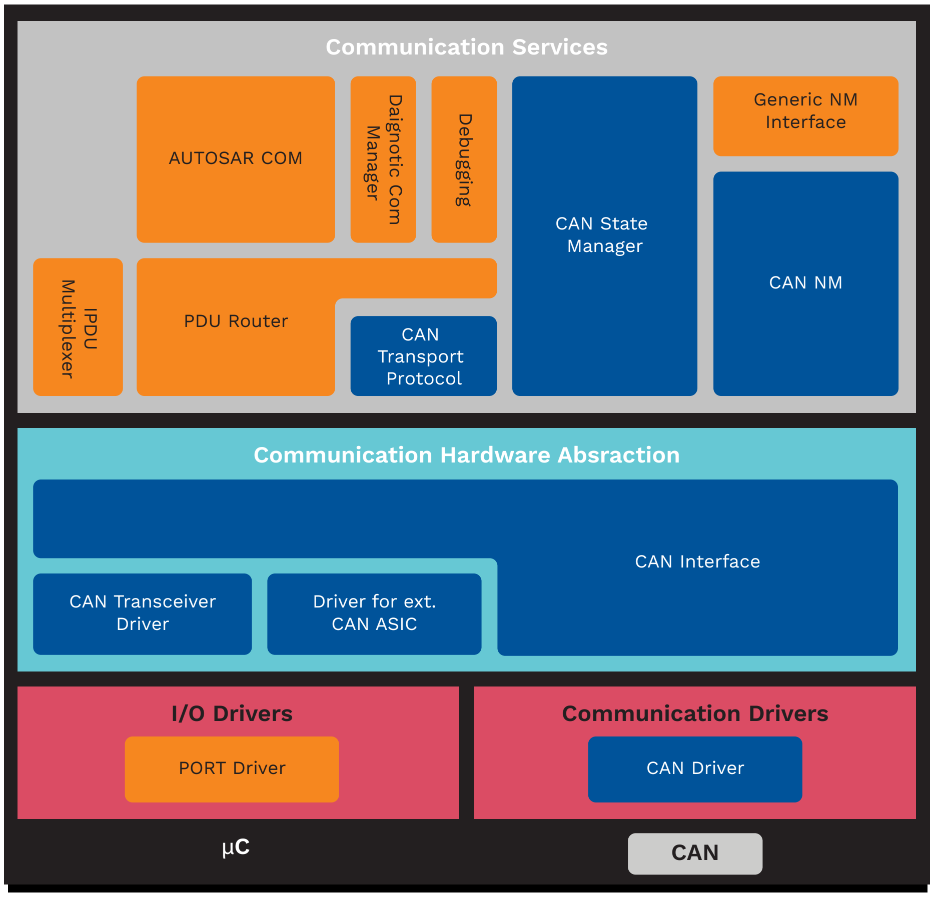


Figure 3: CAN ComStack

Here, we see the protocol/BUS-specific modules are replaced with CAN BUS-specific modules. The following modules are required for the BSW configuration of the CAN Communication stack:

- ECU Configuration (ECUC)
- AUTOSAR COM
- PDU Router (PDUR)
- CAN Interface
- CAN Driver
- Port Driver

3.1. ECUC

ECUC stands for ECU configuration that defines the number of PDUs used in the ECU for communication and it contains the PDU name, data type, and length (DLC). A few basic configuration parameters are mentioned below.

Type	Configuration parameter	Value
PDU	PDU length	Mention in Bytes
	Data Type	UNIT8/ UNIT16/FLOAT32 etc.

Table 1: ECUC Configuration

3.2. AUTOSAR COM

- COM defines four different types of an object for the communication system —I-PDUs, I-PDU Groups, Signals, and Signal Groups.
- Signals in COM are equal to a message/data that contains some information. These signals are grouped together and consistently used for different operations called signal groups.
- Signal groups are used to define the complex data types in AUTOSAR that contains one or more data elements.
- I-PDU is an interaction layer protocol data unit that contains the signals that have been received from an upper layer or a lower layer. The collection of multiple I-PDUs that follow the same direction (either received or transmitted) is called the I-PDU groups.

The basic configuration required is as follows:

Type	Configuration parameter	Value
Com signal	Bit Position	Define the bit position
	Bit Size	Define the bit size (8,16,31 etc)
	Handle Id	It is auto-generated
	Signal type	UNIT8/ UNIT16/FLOAT32 etc.
	Signal reference	PDU ref to signal
COM I-PDU	Direction	Transmit/receive
	Handle ID	It is auto-generated
	I-PDU reference	Gives you a signal reference to the PDU
	I-PDU type	Normal
Transfer Property	I-PDU transfer property	Triggered/Pending/Triggered on change/triggered on change without repetition / triggered on change with repetition

Table 2: Com Configuration

3.3. PDUR

PDUR module is used to route I-PDUs through the CAN communication stack. This configuration is present in the routing table where we define the source and destination I-PDUs.

The basic configuration required is as follows:

Type	Configuration parameter	Value
PDUR BSW module	Upper module	COM/IpduM
	Lower module	CAN IF/CANTP
	Trigger transmit	As per application
Routing table	Routing path for TX/RX PDUs	PDU reference must be given for Source PDU and Destination PDU

Table 3: PDUR Configuration

3.4. CAN Transport Protocol

CANTP is used to segment and reassemble the CAN I-PDUs that are longer than 8-bytes for the standard CAN and 64 bytes for CAN FD.

It also handles TP frames i.e., Single Frame (SF), First Frame (FF), Flow Control (FC), and Consecutive Frame (CF). It receives the I-PDUs from the PDUR and links those I-PDUs to the N-PDUs. It later sends the N-PDUs to the CAN Interface.

3.5. CAN Interface (CanIf)

During the transmission, the CAN Interface receives information from the service layer (PDUR/ CAN TP) and during the reception, from the MCAL (CAN driver). At CanIf, it maps the incoming PDUs from PDUR/TP to Hardware Object Handle (HOH) and vice versa. It has two different hardware object handles for transmission and reception (HRH & HTH) that are mapped to an associated PDU. It receives the I-PDU from PDUR or the N-PDU from CANTP, prepares the L-PDU, and sends it to the CAN driver.

CanIf provides a unique interface to manage different CAN HW, e.g., multiple CAN controllers and CAN transceivers are mapped to a unique interface.

The basic configuration required is as follows:

Type	Configuration parameter	Value
General	Upper module	PDUR/CANTP
	Lower module	CAN driver
	Mode management (Bus control operation and Indication)	Can SM
	Handle type	UNIT8
	No of CAN HW unit	Give the present number of a hardware unit
	CAN ID type	UNIT32
	Error detection	True/false
	Data length check	True/false
	BASIC CAN/ FULL CAN	Select the options (BASIC/Full)
	CAN Id Masking	Add if any specific I Masking/make it free to accept all the messages depending on the requirement.

CAN configuration control	HOH mapping	Reference needs to be given to the associated Hardware object handle to the PDU.
	Driver reference	Driver name reference
	Buffer control	0 if no need for buffer
	Handle mapping to PDU's: HRH: reception HTH: Transmission	Map the transmission PDU's to HTH: Map the reception PDU's HRH:
	Upper layer configuration	PDUR/CANTP Rx indication/ Transmission confirmation
CANIf control	TX and RX L-PDUs	Map the I-PDU/N-PDU to L-PDU
Transmission type	Interrupt/polling	Define as per application

Table 4: CanIf Configuration

3.6. CAN State Manager

CAN SM is used to identify the network in the ECU. An ECU may have different networks and each network has a unique network handle. Each network works in different modes of communication and those modes are requested by the ComM module through CANSm (In Case of CAN). It helps to change the mode of communication depending on the request. It also defines the time duration for the BUS off control.

Type	Configuration parameter	Value
Time duration	Counter for L1 and L2	Depends on requirement (unsigned integer)
	Timer1 L1 (ms)	Depends on requirement (unsigned integer)
	Timer1 L2 (ms)	Depends on requirement (unsigned integer)
	Timer for Tx (ms)	Depends on requirement (unsigned integer)
ComM reference	Network Handle Reference	Give the reference of ComM for specific network of CAN

Table 5: CANSm Configuration

3.7. CAN Driver

It is responsible for initiating transmissions and notifying events when any data is received or given acknowledgment in feedback for data transmission. Multiple CAN controllers can be controlled by one CAN module if they belong to the same CAN hardware unit. (CAN Hardware unit may have more than one CAN controller of the same type and it has one CAN driver).

The basic configuration required is:

Type	Configuration parameter	Value
PDU	PDU to frame reference	PDU reference must be given
	Calculate handle id	True
Baud rate	CAN baud rate configuration	Define the standard baud rate of CAN
CAN Hardware Object	Hardware object for transmission and reception	Mapping of Hardware object handle to hardware object at CAN controller
Clock configuration	Clock reference	Give the clock reference for CAN (mostly from the MCU Module)

Table 6: CAN Driver Configuration

3.8. PORT Driver

It is responsible for the initialization of the port structure of the microcontroller. It defines the direction and functionality of the port pin. The configuration needed for CAN controller pins is: Microcontroller dependent for channel and alternate function selection

- CAN_RX
- CAN_TX

4. Multi-Protocol Communication (LIN/CAN)

If multiple protocols are used, such as CAN and LIN, then, the configuration till the PDU Router will not be affected because till the PDU router, the PDU's are protocol-independent. The application and RTE layer don't have any information about the protocol. After receiving signals from the RTE, AUTOSAR COM will map it to the PDU's and send it to the PDU router. PDU router will decide which protocol (CAN/LIN) that PDU/message will transmit to, by checking its mapping for that PDU and transmit it to protocol-dependent modules, be it Transport protocol (CANTP orLINTP) or to the BUS interface (CANIf/LINIf). In the PDU router, the lower module will change for the PDU's and we need to map its destination and source accordingly (CAN/LIN interface).

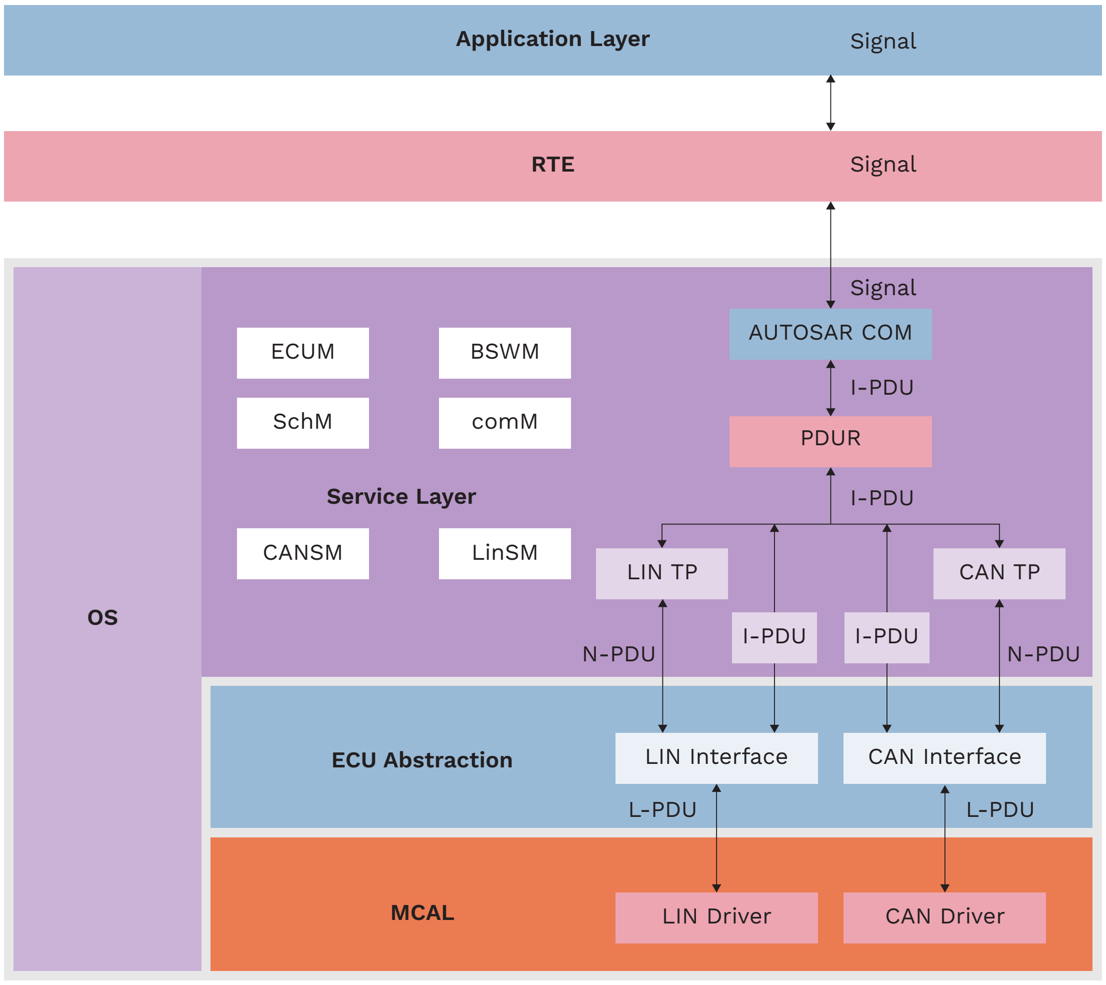


Figure 4: Multi-protocol Stack data flow

During the ECU development, if any changes occur in data/information, we don't need to change anything in the BSW & RTE layer. As per requirement, we need to make changes at the application (SWC) layer, until and unless if you're not adding any new PDU's. If a new PDU is required, you must follow the same procedure (refer sec 3.) for the new PDU's.

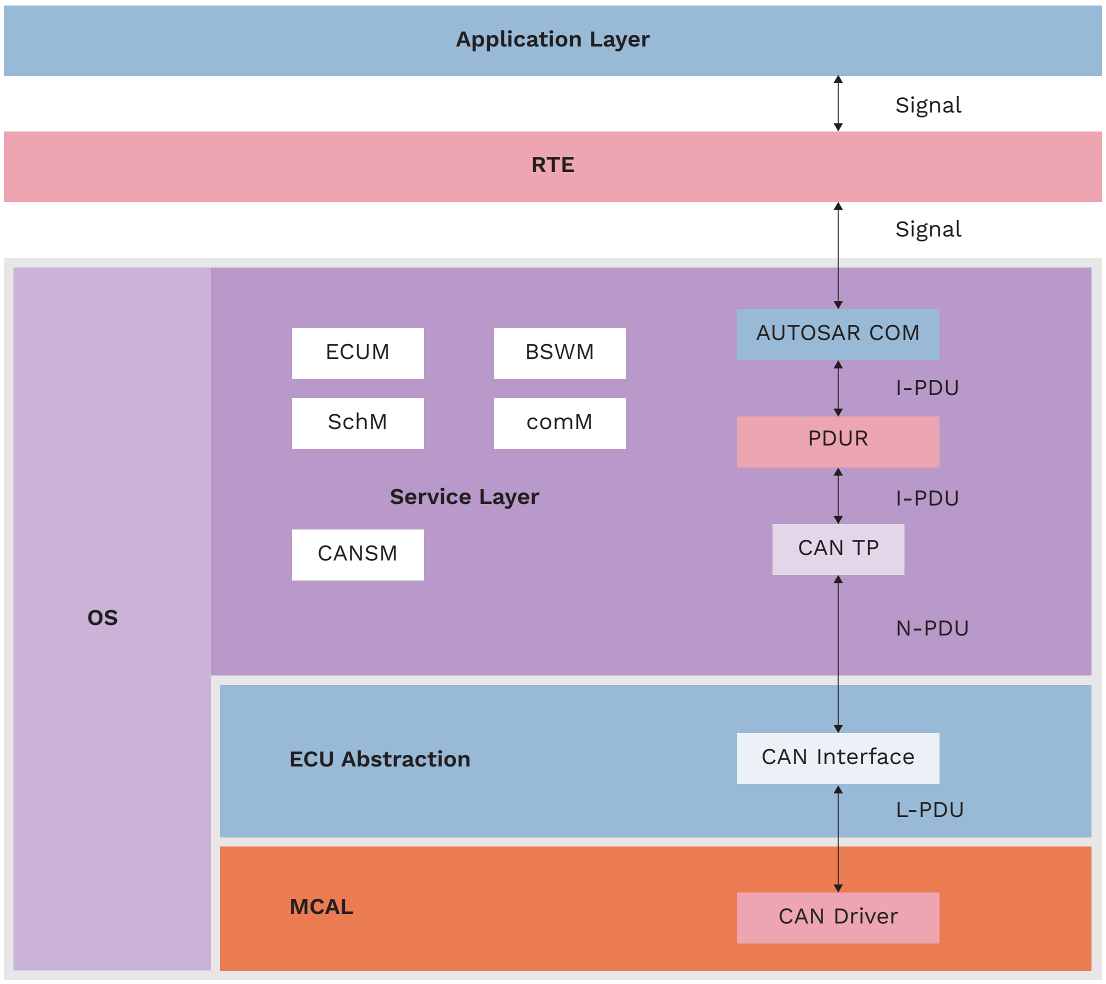
5. Summary

Based on the above-mentioned points, AUTOSAR architecture provides you more flexibility in development that significantly reduces your effort and time. We discussed the significance of AUTOSAR communication stack and the different modules that are required to configure the stack irrespective of any communication protocol (For e.g., LIN, CAN, and Ethernet). We have also tried to explain in detail, the configuration of CAN communication stack and its significance.

6. Appendix

6.1. Protocol Data Unit at each Layer

Protocol data unit is different at each layer while transmitting and receiving data.communication stack and its significance.



6.2. API Interaction

For interaction between different layers and modules, AUTOSAR has defined standard APIs. The diagram below shows API interaction between the layers and modules for CAN ComStack.

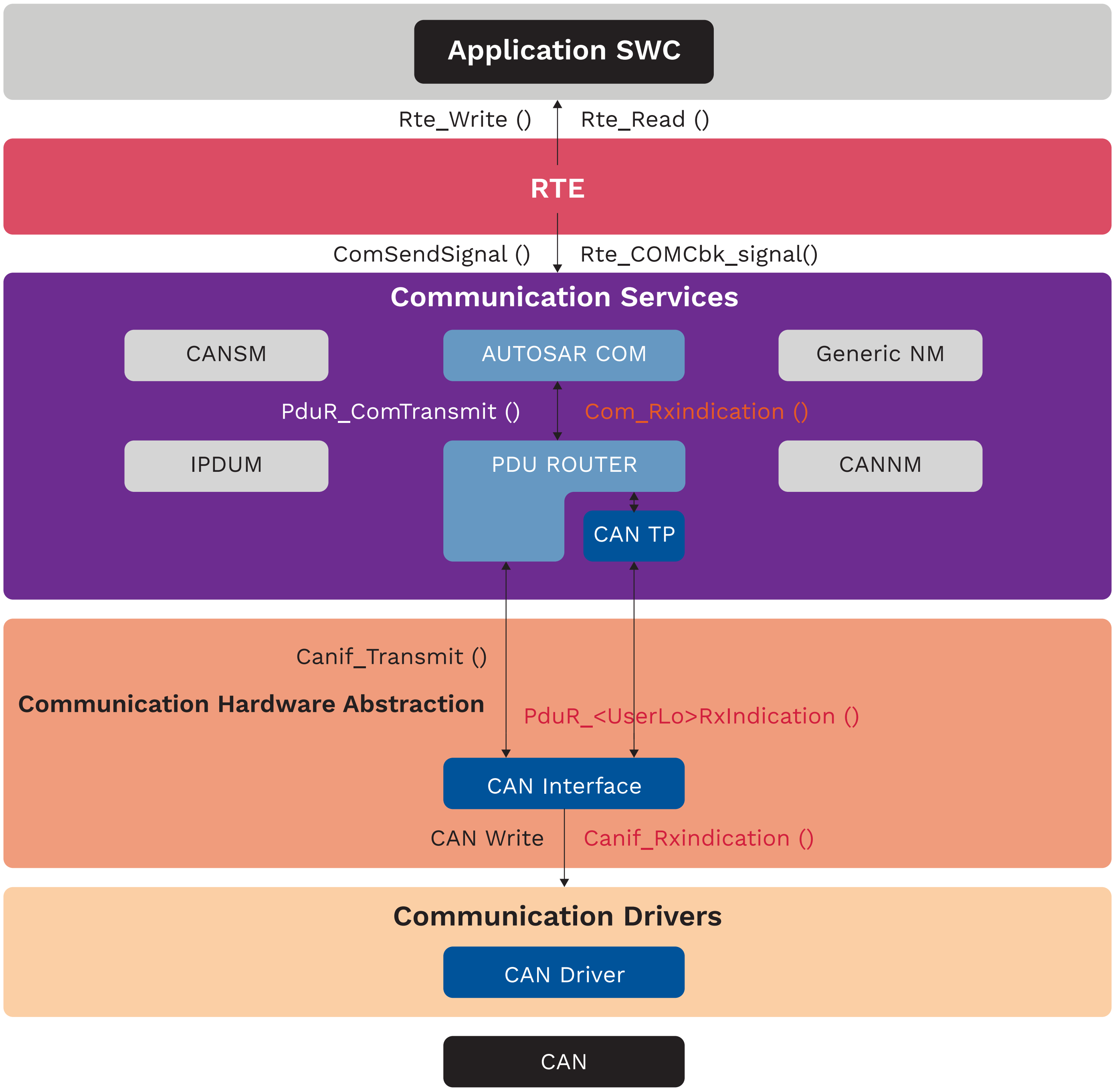


Figure 5: AUTOSAR API Interaction between layers

6.3.1. ECU State Manager (EcuM)

EcuM manages different states of ECUs like initializing and de-initializing the OS, SchM, and BswM as well as some basic software driver modules.

6.3.2. BSW Mode Manager (BswM)

BswM manages the mode requests from application layer SW-Cs or other BSW modules based on simple rules.

6.3.3. Communication Manager Module (ComM)

ComM is a resource manager of COM communication services that controls basic software modules relating to communication.

6.4. Operating System

Task	Scheduling time	Mapped PDU
Task1	10ms	PDU1
Task 2	20ms	PDU2

For the operating system, AUTOSAR refers to OSEK OS [12] (ISO 17356-3). For ComStack OS is used for scheduling and prioritization of tasks. In OS we need to create the task and Schedule them as per requirement. Later, add the PDU's/message into the defined task.

Task1 will trigger automatically every 10ms and send the PDU1.

Note: Configuration parameter names or some additional parameter may be added by tool vendor and those are mostly vendor-specific configuration

About Author

Bhupesh Patil works as an Engineer at eInfochips. He majorly works in the automotive domain including AUTOSAR and Non-AUTOSAR. In AUTOSAR he mostly worked on Communication stack (CAN) whereas other experience includes UDS, UDS bootloader, Sensors calculation/calibration, firmware development, and micro-controllers. He holds a Bachelor of Engineering Degree in Electronics and Telecommunications and a Post graduate diploma in Embedded System.

eInfochips is a CMMi Level 3, ASPICE Level 2, ISO 9001:2008, and ISO 26262 safety standard certified Product Engineering Services company. We provide the AUTOSAR architecture implementation and integration facilities such as RTE, BSW configuration, OS, MCAL integration, and SWC (application) development compliant with ISO26262- Safety standard on various platforms (For e.g., NXP, Infineon, Renesas, and SPC5). We also provide services in the functional and integration testing of AUTOSAR compliance products.

References

AUTOSAR standards document: https://www.autosar.org/standards/classic-platform/classic-platform-431/