Document Title	Complex Driver design and integration guideline
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	622

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R25-11

Document Change History

Date	Release	Changed by	Description
2025-11-27	R25-11	AUTOSAR Release Management	• Clarifications regarding the StbM.
2024-11-27	R24-11	AUTOSAR Release Management	• Clarifications regarding names of include files.
2023-11-23	R23-11	AUTOSAR Release Management	• No content changes
2022-11-24	R22-11	AUTOSAR Release Management	• Update of the figure which describes the header files hierarchy of a CDD module
2021-11-25	R21-11	AUTOSAR Release Management	• No content changes
2020-11-30	R20-11	AUTOSAR Release Management	• Update for Module IDs for non-AUTOSAR BSW modules
2019-11-28	R19-11	AUTOSAR Release Management	• Add a note in the 7.3.8 chapter • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Remove SWS_EcuMfixed in Chapters 4.1 and 7.3.2
2017-12-08	4.3.1	AUTOSAR Release Management	• Adapt the 7.3.9 chapter title

△

2016-11-30	4.3.0	AUTOSAR Release Management	• Add chapter to interface with StbM module • Update for Module ID
2015-07-31	4.2.2	AUTOSAR Release Management	• Update for Default Error Tracer • Re-entrancy of interfaces
2014-10-31	4.2.1	AUTOSAR Release Management	• Update for TcpIp
2014-03-31	4.1.3	AUTOSAR Administration	• Update of CDD code files chapter • Removed chapters on change documentation
2013-03-15	4.1.1	AUTOSAR Administration	• Initial release

Disclaimer

Table of Contents

1 Introduction

1.1 Scope of Document

The purpose of this document is to:

- Give an overview of Complex Driver (CDD)
- Give recommendations for implementation and integration of a CDD within AUTOSAR architecture.

This document is aimed at developers and integrators of CDD.

2 Definition of terms and acronyms

There are no acronyms and abbreviations, which have a local scope. The AUTOSAR glossary [1] contains all acronyms and abbreviations used in this document.

3 Conventions to be used

No content.

4 Related Documentation

4.1 Input documents & related standards and norms

[1] Glossary
AUTOSAR_FO_TR_Glossary

[2] Layered Software Architecture
AUTOSAR_CP_EXP_LayeredSoftwareArchitecture

[3] General Specification of Basic Software Modules
AUTOSAR_CP_SWS_BSWGeneral

[4] General Requirements on Basic Software Modules
AUTOSAR_CP_RS_BSWGeneral

[5] Specification of Standard Types
AUTOSAR_CP_SWS_StandardTypes

[6] Specification of Platform Types for Classic Platform
AUTOSAR_CP_SWS_PlatformTypes

[7] Specification of Communication Stack Types
AUTOSAR_CP_SWS_CommunicationStackTypes

[8] Basic Software Module Description Template
AUTOSAR_CP_TPS_BSWModuleDescriptionTemplate

[9] Specification of ECU Configuration
AUTOSAR_CP_TPS_ECUConfiguration

[10] Software Component Template
AUTOSAR_CP_TPS_SoftwareComponentTemplate

[11] Specification of ECU State Manager
AUTOSAR_CP_SWS_ECUStateManager

[12] Specification of Watchdog Manager
AUTOSAR_CP_SWS_WatchdogManager

[13] Specification of PDU Router
AUTOSAR_CP_SWS_PDURouter

[14] Specification of Communication
AUTOSAR_CP_SWS_COM

[15] Specification of Communication Manager
AUTOSAR_CP_SWS_COMManager

[16] Specification of Network Management Interface
AUTOSAR_CP_SWS_NetworkManagementInterface

[17] Specification of TCP/IP Stack

AUTOSAR_CP_SWS_TcpIp

[18] Description of the AUTOSAR standard errors
AUTOSAR_CP_EXP_ErrorDescription

[19] Specification of Operating System
AUTOSAR_CP_SWS_OS

[20] Specification of Synchronized Time-Base Manager
AUTOSAR_CP_SWS_SynchronizedTimeBaseManager

5 Introduction to CDD

A Complex Driver is a software entity not standardized by AUTOSAR that can access or be accessed via AUTOSAR Interfaces and/or Basic Software Modules APIs.

According to the document [2] *Layered Software Architecture*, a CDD is a specific module located in the Complex Drivers Layer of the Basic SoftWare which interacts with standard BSW modules or Rte.

- A CDD may need to interface to modules of the layered software architecture
- A module of the layered software architecture may need to interface to a CDD
- A CDD may need to interface SW-Cs via Rte

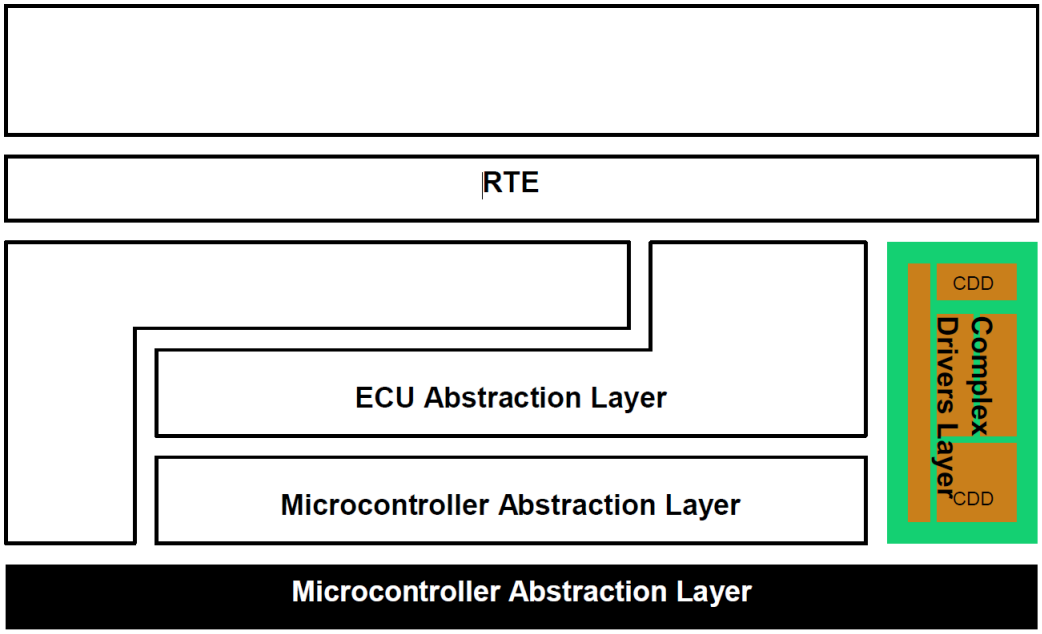

Figure 5.1: CDD in Layered Software Architecture

The main goal of the CDD is to implement complex sensor evaluation and actuator control with direct access to the microcontroller using specific interrupts and/or complex microcontroller peripherals, external devices (communication transceivers, ASIC...) to fulfill the special functional and timing requirements.

In addition it might be used to implement enhanced services / protocols or encapsulates legacy functionality of a non-AUTOSAR system.

CDD Implementation might be application, Microcontroler and ECU dependent.

Finally, the CDD can serve as mechanism of migration to introduce existing or new concepts into the AUTOSAR Software Architecture.

6 CDD design recommendations

To interface and ease CDD integration in AUTOSAR architecture, the designer shall take into consideration the following points.

6.1 Documentations

6.1.1 User's Manual

CDD designer shall provide a User's Manual to ease the integration and provide information to customers:

- CDD introduction and overview
- Description of the functional operations (initialisation, normal, shutdown, fault operation...)
- Description of the relationship with and need from other BSW Modules, SchM and Rte; e.g. memory blocks from NvM, critical sections to configure.
- Files structure and dependencies
- Description of the interfaces (including services): name, description, re-entrancy, parameters (names, types, ranges, values), return value (name, type, range, values), configuration class.
- Description of the non-functional requirements: timing and behaviour requirements, resource usage, behaviour with other BSW modules or SW-C...
- Description of the Dem errors, optionally Det errors, debug variables
- Description of the configuration parameters (names, types, ranges, values).
- Description of the memory mapping needs (Flash, RAM)
- Usage limitations and open issues
- Integration constraints and requirements to other modules
- Examples

6.1.1.1 Module ID

The range of possible module IDs for CDDs is described in the document [3] *General Specification of Basic Software Modules*.

6.2 Implementation

There are few constraints coming from AUTOSAR regarding CDD implementation. At least:

- CDD shall respect the input specifications [4], [2], [3], [5], [6], [7], [8], [9].
- CDD shall protect its critical resources defining critical sections which can be handled by SchM or OS mechanisms.
- CDD mode may be manageable by EcuM and BswM modules.
- CDD may handle its memory sections using the memory mapping mechanisms.
- CDD may report its errors using Det or Dem modules.

6.3 CDD Files

This section is only a recommendation and does not completely define the module files structure.

6.3.1 Code file(s)

The code file structure of the CDD module is not fixed, beside the requirements in the document [4] *General Requirements on Basic Software Modules* and the document [3] *General Specification on Basic Software Modules*.

At least, a **CDD_<MODULENAME>.c** shall be provided.

Interrupt functions may be placed in a **CDD_<MODULENAME>_Irq.c**.

Callout functions may be placed in a **CDD_<MODULENAME>_Callout.c**.

Depending of the need, C objects generated at Link time from configuration may be placed in **CDD_<MODULENAME>_Lcfg.c** file.

Depending of the need, C objects generated at Post Build time from configuration may be placed in **CDD_<MODULENAME>_PBcfg.c** file.

If an implementation of the CDD module requires additional code files, it is free to include them.

6.3.2 Header file(s)

The following figure contains the defined AUTOSAR header file hierarchy of the CDD module.

CDD module shall provide a header file structure, so that users of the CDD module needs only to include the **CDD_<MODULENAME>.h** file.

Depending of the need, C objects declarations generated from configuration may be placed in **CDD_<MODULENAME>_Cfg.h**, **CDD_<MODULENAME>_PBcfg.h**, **CDD_<MODULENAME>_Lcfg.h** files.

If an implementation of the CDD module requires additional header files, it is free to include them. The header files are self contained, that means they will include all other header files which are required by them.

CDD module may include **Det.h** and/or **Dem.h** header files to report errors.

CDD module may include **<Mip>_MemMap.h** header file if some memory mapping area have to be defined where <Mip> is the Module Implementation Prefix.

CDD module may include **Rte_<CDD_MODULENAME>.h** header file if interfaces to the Rte are configured.

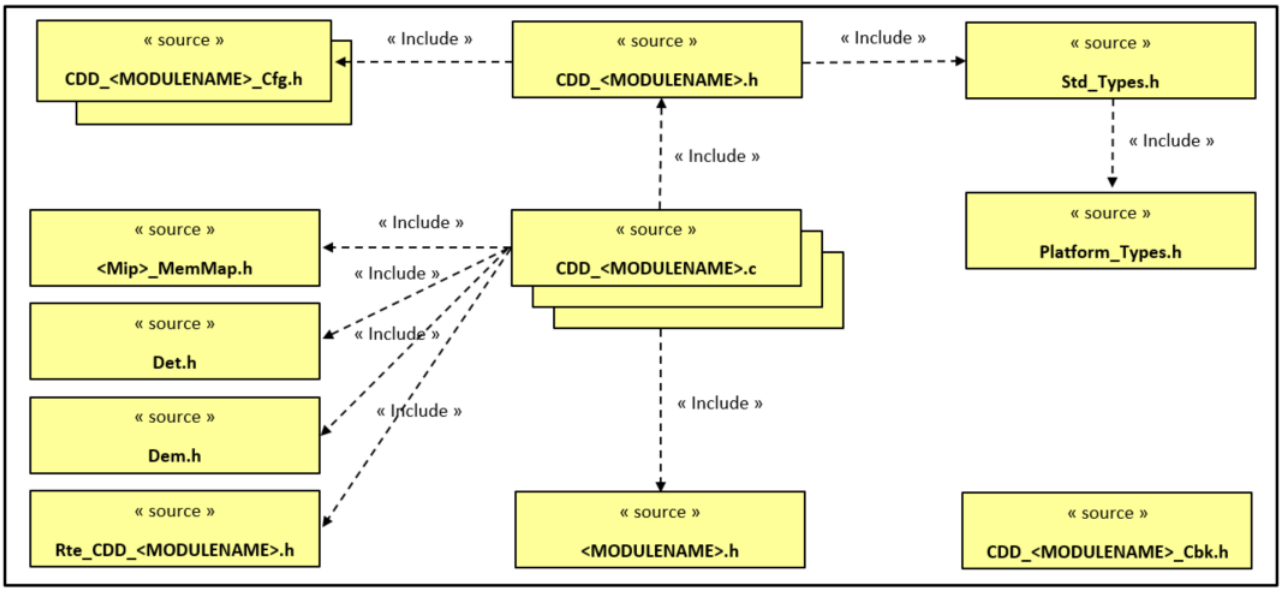

Figure 6.1: Header File Structure for CDD

6.3.3 Coherence checks

The CDD module shall avoid the integration of incompatible (.c or .h) files as defined in the document [3] *General Specification on Basic Software Modules*.

6.4 Behaviour and Interfaces description

Some CDD not only have interfaces to other BSW modules or clusters, but have also more abstract interfaces accessed from application SW-Cs via the Rte.

In these cases a CDD SW-C type is needed to interface the Rte and CDD shall respect the requirements of the document [8] *Basic Software Module Description Template* and [10] *Software Component Template*.

This description file should contain:

- Description of the CDD services
- Types and ports interfaces
- Description of internal behaviour and runnable Entities
- Description of the required triggered events of runnable Entities
- Description of exclusive Areas for shared resources protection
- Memory mapping

The more abstract interfaces required here are called AUTOSAR Interfaces which are described by means of the Software Component Template (SWCT), they consist of ports, port interfaces and their further detailing.

The root classes of the SWCT used to describe these elements for CDD are `ComplexDeviceDriverSwComponentType`.

The function calls from the Rte into these CDD shall be modelled as `RunnableEntities` which are also contained in the SWCT. The root class of the SWCT used to describe the `RunnableEntities` (and a few other things) is called `SwcInternalBehavior`.

Hints: CDD runnables should be designed to reduce Rte overhead, e.g.

- Server runnables is prefered to be re-entrant: `can be invoked concurently = TRUE`.
- Runnables signature to be: `void` or `StdReturnType RunnableName(void or parameters)`

6.5 Parameters configuration

If parameters has to be configured using an AUTOSAR GCE, CDD shall respect the requirements of the document [9] *Specification of ECU Configuration*.

At Least:

- AUTOSAR and Software versions of the modules shall be identified by the configuration file.
- Det should not be included for production phase, so a parameter is needed in the configuration to deactivate the error report.

7 Interfacing to other modules

This section describes the relations to other modules within the basic software.

7.1 Interfacing to Rte and SW-C

CDD may need to interface to SW-Cs through the Rte:

- Required ports and interfaces shall be specified and implemented according to AUTOSAR (AUTOSAR interface).
- In some cases, CDD has to use some port specific parameters defined by Rte.

Refer to previous chapter 6.4 .

7.2 Interfacing to libraries

CDD can use AUTOSAR libraries.

Example: CDD can use E2E library mechanism to transmit the communication protections against a corruption or a loss of data.

7.3 Interfacing to standard BSW modules

CDD may need to interface to other modules in the layered software architecture, or modules in the layered software architecture may need to interface to a CDD. If this is the case, the following recommendations apply:

Interfacing from modules of the layered software architecture to CDD:

CDD shall offer interface(s) which can be generically configured by the accessing AUTOSAR module.

A typical example is the PDU Router: a CDD may implement the interface module of a new bus system. This is already taken care of within the configuration of the PDU Router.

Interfacing from a CDD to modules of the layered software architecture:

This is only allowed if the respective modules of the layered software architecture offer the interfaces, and are prepared to be accessed by a CDD. Usually this means that:

- CDD shall take care of re-entrancy of interfaces. For non-re-entrant interfaces only one caller can access the interface. For conditionally re-entrant interfaces several callers may access the interface concurrently if they use different IDs.
- If call back routines are used, the names are configurable

- No upper module exists which does a management of states of the module (parallel access would change states without being noticed by the upper module)

CDD shall provide all configuration parameters which are necessary to satisfy other AUTOSAR modules which rely on the information, e.g. if Dem is called to report production errors, the Dem error codes must be defined and referenced inside the CDD configuration in line with the configuration standard for Dem error code definition.

In case of multi core architectures, refer to chapter 7.4 .

In general, it is possible to access the following modules:

7.3.1 Interfacing with MCAL modules

CDD may directly access to microcontroller resources (e.g. a hardware timer). CDD should use the MCAL if the needed resource is managed by a MCAL module and if there are no specific constraints (e.g. real time need). This is highly recommended to avoid conflicts (e.g. Parallel access to the same group/channel/etc. is mostly not allowed because DIO services are not re-entrant).

In this case, CDD shall use the standard API of the MCAL modules to access MCAL modules.

7.3.2 Interfacing with BSW Mode Manager & ECU State Manager

The EcuM and the BSW Mode Manager shall be the exclusive access points to the mode management in case the ECU State Manager is used.

ECU State Manager should be used for:

- Init and De-Init functions shall be exclusively called by the EcuM and/or the BswM modules.
- If a CDD handles a wakeup source, it must follow the protocol for handling wakeup events specified in the document [11] *Specification of ECU State Manager*.

BSW Mode Manager should be used for:

- CDD modes changed management
- The BswM (which is on the master core) ascertains that the ECU should be shutdown and distributes an appropriate mode switch to each core. The CDD on the slave cores must catch this mode switch, de-initialize appropriately and send appropriate signals to the BswM to indicate their readiness.

7.3.3 Interfacing with Memory Stack

Direct access outside NVRAM manager is possible if it is exclusively managed by CDD. If CDD uses the standard memory stack, the NVRAM Manager is the exclusive access point to the memory stack: CDD shall use the NVM's API to access memory.

7.3.4 Interfacing with Watchdog Stack

The Watchdog manager may supervise the execution of one or more runnables of a CDD as supervised entities. The watchdog manager shall be configured and CDD runnable shall call Watchdog API as described in the document [12] *Specification of Watchdog Manager*.

The Watchdog Manager is the exclusive access point to the watchdog stack.

CDD should not interact directly with the watchdog manager but through the Rte defined ports.

Usually, the Rte is responsible for propagating Checkpoint information from Supervised Entities in CDD to the Watchdog Manager module. The Watchdog Manager module uses the services of the Runtime Environment to inform CDD about changes in the supervision status.

To control the state-dependent behavior of CDD, the Rte provides the mechanism of mode ports. A mode manager can switch between different modes that are defined in the mode port. The CDD that connects to the mode port can use the mode information in two ways:

- The CDD can query the current mode via the mode port.
- The CDD can declare Runnables that are started or stopped by the Rte because of mode changes.

In case of failure, the Watchdog Manager may inform the CDD Supervised Entity about supervision failures via the Rte Mode mechanism. The CDD Supervised Entity may then take its actions to recover from that failure.

7.3.5 Interfacing with Communication Stack

Several access points are possible:

- It is possible to interface to the PDU Router module to handle IPDU.
- It is possible to interface to the <Bus> Interface.
- It is possible to interface to the NM module.
- It is possible to interface the Tcpip module.

- It is possible to directly interface to Com module as it is possible to have signal interface.

Generally, it is not suitable to mix the access points, i.e. use PduR access at the same time as Com access or <Bus> Interface.

CDD which handles communication and may trigger transmission of PDUs should provide an API to enable/disable transmission. This will e.g. enable the Dcm to disable the whole communication in a corresponding diagnostic request. These functions provided by the CDD may be called in the configured action list which is linked to this function. For example to these functions, refer to similar API within the communication stack.

7.3.5.1 Interfacing with PDU Router

The PduR is the exclusive bus and protocol independent access point to the communication stack for IPDU.

CDD shall use standard APIs of the PduR module to access IPDU.

When CDD Interacts with the PduR, one container per CDD shall be configured within the PduR.

Refer to the document [13] *Specification of PDU Router* to get more details.

7.3.5.2 Interfacing <Bus> Interfaces modules

The <Bus> Interfaces modules are the exclusive bus specific access point to the communication stack.

CDD shall use standard APIs of the <Bus> Interfaces modules to access IPDU.

When CDD interacts with <Bus> Interface, CDD uses the access functions defined for <Bus> Interface and <Bus> Interface callbacks shall be configured according to CDD needs.

Refer to <BUS> Interface specifications and user's manual for details.

7.3.5.3 Interfacing with Com Module

If CDD handles Com signals, CDD shall use standard APIs of the Com module or Rte define to access signals.

Refer to the document [14] *Specification of Communication* to get more details.

7.3.5.4 Interfacing with Com Manager

If CDD uses Com signals, CDD shall use standard APIs of the Com Manager to request a "Communication Mode".

If CDD handles a <Bus> which is not AUTOSAR Standard, <Bus> States should be handled by ComM to coordinate bus communication stack.

Refer to the document [15] *Specification of Communication Manager* to get more details.

7.3.5.5 Interfacing with Network Management Interface module

If CDD handle a <Bus> which is not AUTOSAR Standard, <Bus> States should be handled by a `<Bus>Nm_CDD` module.

The `<Bus>Nm_CDD` should provide services to the Network Manager to manage <Bus> States.

Refer to the document [16] *Specification of Network Management Interface* to get more details.

7.3.5.6 Interfacing with TCP/IP module

The TcpIp module is the exclusive socket-based access point to the communication stack.

CDD shall use standard APIs of the TCP/IP module to access sockets.

Refer to the document [17] *Specification of TCP/IP Stack* to get more details.

7.3.6 Interfacing with XCP module

If CDD handle a <Bus> which is not AUTOSAR Standard, XCP can interface `<Bus>_CDD` to forward the data.

The XCP module offers configurable interfaces to be used by CDD:

- `<Cdd_Transmit>` : API to request the sending of a PDU via CDD
- `<Xcp_CddTxConfirmation>` : API to confirm the successful transmission of the PDU
- `<Xcp_CddRxIndication>` : API API called by the CDD to indicate a successful reception of a LPDU.

The XCP module shall be configured to allow CDD functionalities: `XcpOnCddEnabled` parameter shall be activated.

If needed, CDD may call callback function `Xcp_<module>RxIndication`.

7.3.7 Interfacing with Diagnostic Log and Trace

If CDD handle a <Bus> which is not AUTOSAR Standard, Dlt can interface `<Bus>_CDD` to forward the data.

The Dlt forwards the data to the Dcm or a CDD which uses a serial interface for example.

Dlt does not define a specific communication interface. The Dlt specification defines an API to an internal Dlt communication module. It is up to the implementer, how this communication module is implemented and how it communicates with a possible CDD (e.g. Serial or USB).

7.3.8 Interfacing with Default Error Tracer and Diagnostic Event Manager

CDD shall report errors using Det, Dem as described in the document [18] *Description of the AUTOSAR standard errors*.

CDD shall use standard APIs of the Det & Dem modules.

CDD shall react as any BSW modules. Error ID shall be defined locally in the CDD module. CDD is responsible for initiating an internal recovery.

Note: For calls to the Det the module id and/or the instance id parameter can be used to distinguish between different CDDs.

7.3.9 Interfacing with OS

Usually, only the BSW Scheduler and the Rte shall use OS objects or OS services. Therefore, the CDD should only access to `GetCounterValue` and `GetElapsedCounterValue` services of the OS.

The OS can be accessed by CDD as long as the used OS objects are not used by another BSW module, e.g. CDD could create an OS alarm and use it.

OS can notify CDD by `OsRestartTask` that an OS-Application has been terminated and restarted. The CDD will then have to take appropriate clean-up actions.

Refer to the document [19] *Specification of Operating System* to get more details.

7.3.10 Interfacing with StbM module

If a CDD module implements a user defined Timebase Provider, i.e., if it handles Global Time Synchronization messages, the CDD module shall use the StbM module API:

- `StbM_GetCurrentTime` to read latest time base value from StbM
- `StbM_GetCurrentVirtualLocalTime` to calculate time base value updates
- `StbM_BusSetGlobalTime` to forward time base values received on the bus to StbM

Refer to the document [20] *Specification of Synchronized Time Base Manager* for details about the API.

Relevant details of the configuration of the CDD module for Global Time Synchronization are specified by the container `CddGlobalTimeContribution` in the CDD's module definition. Refer to the document [9] *Specification of the ECU Configuration*.

7.4 CDD in multi-cores system

CDD can be used in multi-cores architecture.

In case of multi core architectures, CDD can reside on any core(s) respecting the following rules:

- Crossing partition and core boundaries is permitted for module

  internal communication only, using a master/satellite implementation.
- Consequently, if the CDD needs to access standardized interfaces of the BSW, it needs to reside on the same core.
- In case a CDD resides on a different core, it can use the normal port mechanism to access AUTOSAR interfaces and standardized AUTOSAR interfaces. This invokes the Rte, which uses the IOC mechanism of the operating system to transfer requests to the other core.
- However, if the CDD needs to access standardized interfaces of the BSW and does not reside on the same core,
  - either a satellite providing the standardized interface can run on the core where the CDD resides and forward the call to the other core
  - or a stub part of the CDD needs to be implemented on the other core, and communication needs to be organized CDD-local using the IOC mechanism of the operating system similar to what the Rte does.
- Additionally, in the latter case the initialization part of the CDD also needs to reside in the stub part on the different core.

7.5 CDD module of the MCAL

CDD for microcontroller driver can be performed but it cannot access to other standard module as it is in the lower layer except Det, Dem, SchM...

In general, if some limitations are applied to a specific layer, it applies also to CDD.