ANCIT AUTOSAR REFERENCE CARD

A QUICK GUIDE TO REFER FOR AUTOSAR STACKS

ANCIT EDUTECH

ANCIT CONSULTING

COIMBATORE
BANGALORE
DINDUGAL

beeshma@ancitconsulting.com
info@ancitconsulting.com

9840378602
9843541953

For AUTOSAR related technical videos

https://www.youtube.com/@ancittechnicalvideos

AUTOSAR INTRO

WHY AUTOSAR

To promote an open and standardized software architecture for automotive ECUs. It

- improves ECU software quality,
- reduces development costs
- avoid re-development of similar ECU software components repeatedly for the same vehicular application
- isolates the application from the underlying software

Ref:- https://www.youtube.com/watch?v=NfZI8wvgZPo

AUTOSAR ARCHITECTURE

➡ **APPLICATION LAYER**

The application Layer contain the software components (SWCs), which realize the application functionality of the ECU

➡ **RUN TIME ENVIRONMENT (RTE)**

- RTE establishes communication between SWCs,
- Scheduling of SWCs.

➡ **SERVICE LAYER**

Responsible for providing services independent of underlying hardware or protocol. Includes OS, Memory Manager, ECUM, Diagnostic Services

➡ **ECU ABSTRACTION LAYER**

It makes higher software layer independent of ECU Hardware Layout.

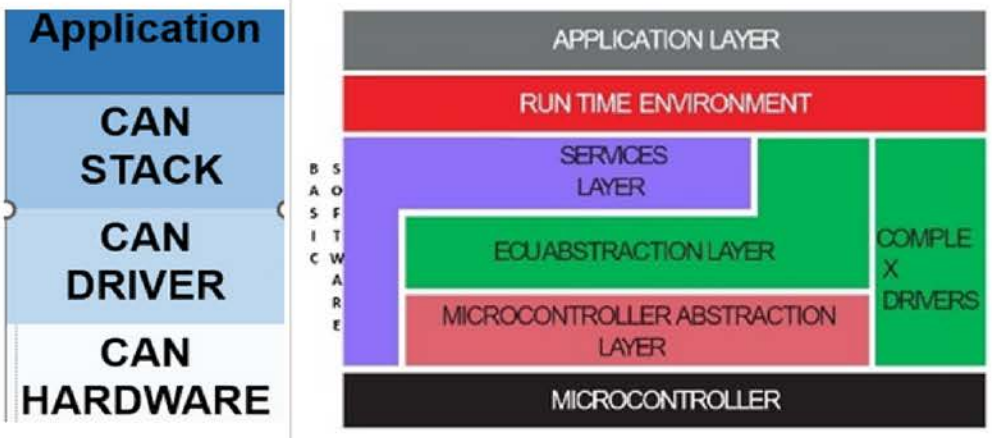
➡ **MICROCONTROLLER ABSTRACTION LAYER**

Contains the internal drivers to access to the microcontroller internal peripherals.

➡ **COMPLEX DRIVERS**

The Complex Drivers spans from the Hardware to the RTE. SWCs which do not follow AUTOSAR standard are integrated into the system through CDD

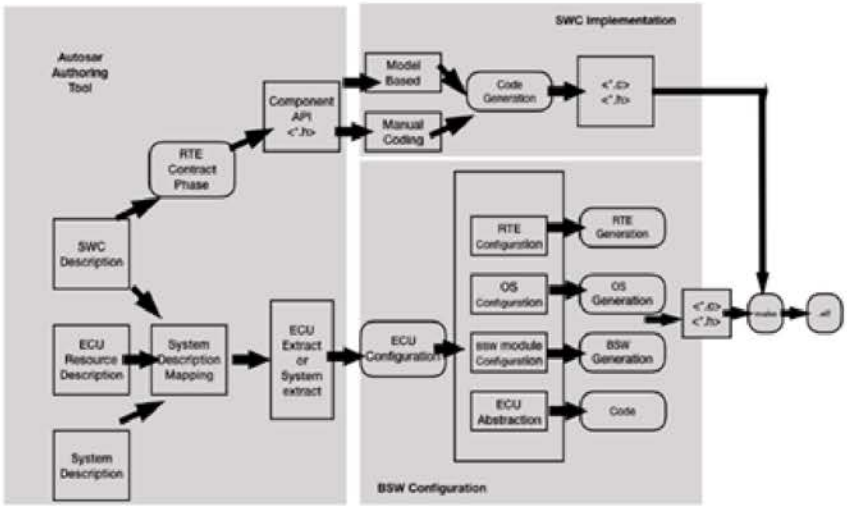
AUTOSAR ARCHITECTURE



Legacy Software Architecture has only two layers.
The CAN stack is usually bought for specific series of boards.

AUTOSAR has multiple layers.

AUTOSAR METHODOLOGY



Autosar Methodology defines the sequence of steps/ activities to develop an AUTOSAR system

Ref:- *https://www.autosar.org/standards/classic-platform*

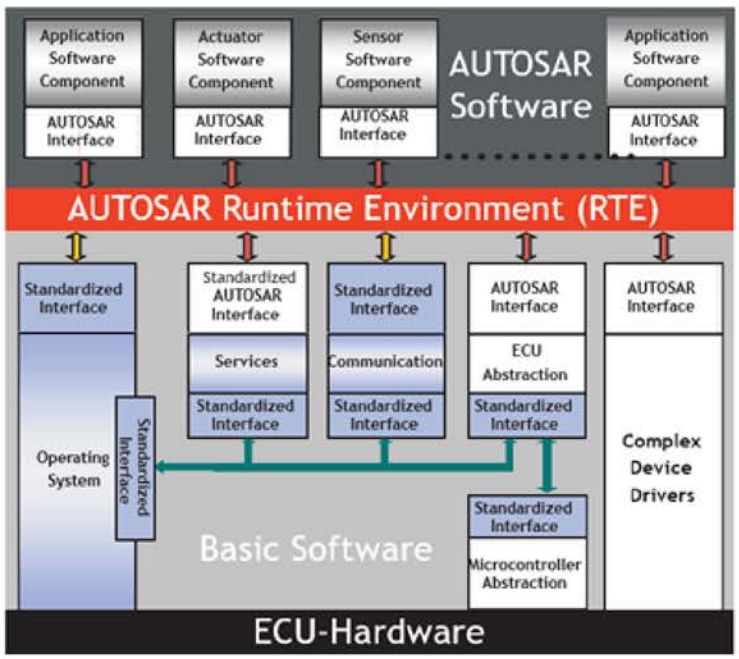
VFB DESIGN

The VFB "virtual functional bus" is the Sofwtare Design of the system.
Ref:- https://autosartutorials.com/virtual-function-bus-vfb-concept/

SYSTEM EXTRACT AND ECU EXTRACT

- Ecu Extract - contains signals and messages to a single ECU
- System Extract –contains signals and messages information related to the system (cluster).

TYPES OF AUTOSAR INTERFACE



- **Autosar Interface**- The function and function argument is user defined.
  Eg:- Rte_Write_ledDial_ledData(seatSensor.seatStatus);

- **Standardized Interface**- The function name is standard but the argument is user defined
  Eg:-Com_SendSignal(ComConf_ComSignal_CanTxSignal, &value);

- **Autosar Standardized Interface**- Whose syntax and semantic are Standardized in Autosar.
  *Eg:- Std_ReturnType BswM_PduGroupSwitchActionHandle(const BswM ActionListItemType *item);*

Refer the picture to check where the interfaces are used

APPLICATION LAYER

TYPES OF SWC

Application, Sensor Actuator, Complex Device Driver, NV Block Service , ECU abstraction , Atomic SWcs

AUTOSAR provides us the flexibility to break an application software *(ex: airbag)* into smaller components. The smallest software component is called Atomic component.

DATA TYPES

Base DataType ➡ It's a basic Datatype that used in Embedded Systems *Eg:- Uint8_t, Uint16_t*

Implementation DataType(IDT) ➡ Its is the datatype that used in AUTOSAR Code/C code (temperature). Its always mapped with base Datatype

Application DataType(ADT) ➡ This datatype is mostly used in Application Layer (Degrees) and its mapped with IDT. Used only for reader to understand.

TYPES OF COMMUNICATION(C\S AND S\R)

SENDER \ RECEIVER INTERFACE Sender Receiver in AUTOSAR is used to exchange (Read or Write) Data from one software component to another software component /Service component(BSW). *Similar to extern keyword usage in C*

CLIENT\ SERVER INTERFACE Client\Server interface defines the Function prototype that can be invoked based on the client server ports *(Function call and function definition in C)*

OTHER INTERESTING INTERFACES

MODE SWITCH INTERFACE Used to inform the application about a change of mode (*ex: ComM mode changes from no Com to Full Com)*

TYPES OF PORTS

Each interface has ports associated with it.

APPLICATION PORTS Ports Created by users. The ports use AUTOSAR Interface.

SERVICE PORTS Ports are AUTOSAR defined and these ports use Standardized AUTOSAR interface. *Ref:- https://www.ancitedutech.com/blog/ports-application-ports-service-ports-and-calibration-ports/*

RUNNABLES

Runnable entity is a part of SWC where the application behaviour logic is written. Runnable is analogous to functions in C.

ASWC DESIGN

```
CtSaFrontDoor.c  //Atomic Software Component (Sensor Actuator component)
RunnableDoorStatus() // This is the Runnable that is called by Rte as per the corresponding Event
{
IDTDoorStatus DoorStatus; // Local variable declaration with Implementation datatype
GetDoorStatus(&DoorStatus) // This is the Client API to read the value of Door Status from CDD
SendDoorStatus(DoorStatus) // This is the Sender API to Provide value to CtApAlogorithm.c File
}

CtApAlgorithm.c  // Atomic software Component (application  Component)
RunnableDoorReceive() // This is the Runnable that called by Rte as per the corresponding Event
{
 IDTDoorStatus DoorStatus1; // Local variable declaration with Implementation datatype
ReseiveDoorStatus(DoorStatus1) // This is the ReceiverAPI to take value from  CtApAlogorithm.c File
}



CDD.C // Complex drivers software Component
GetDoorStatus(IDTDoorStatus *DoorStatus) // This is the Server for providing the value to cleint
{
// Write MCAL Api to Access the drivers and Read the Door Status
}
NOTE: the APIs used inside the runnables will not be generated by code; The AUTOSAR tool will only
generate the API interface. User has to use it inside the runnable
```

RTE RESPONSIBILITIES

- Acts as a middleware between the AUTOSAR application layer and the lower layers.
- Responsible for establishing communication between Application SWCs and BSW modules
- Responsible for scheduling the runnables. The runnables are triggered based on RTE Events

RTE CONTRACT PHASE VS RTE GENERATION PHASE

RTE CONTRACT PHASE

- In this phase only limited information about the component is available
- So when a software-component is delivered it shall consist of the following parts:
    1. SW-Component Type Description
    2. SW-Component Internal Behavior Description
    3. The actual SW-Component implementation and/or compiled SW-Component
    4. SW-Component Implementation Description
    5. APIs to access the Ports and send and receive data.

RTE GENERATION PHASE

- In this phase all the relevant information about components, deployment to ECU, communication connections of the ECU is defined and the information is used to generate the RTE. One RTE is generated for each ECU in the system
- The RTE Generator in the Generation Phase is also responsible to generate
    1. Details about the Basic Software Module Description generation
    2. Details about the MC-Support
    3. IOC configurations and uses an implementation specific deterministic generation scheme
    4. The corresponding C data types which will be generated into the Rte_Type.h.

Ref: https://www.linkedin.com/pulse/autosar-rte-generation-kaarthick-balakrishnan/

RTE CONFIGURATION - WORK DONE BY USER AND TOOL

WORK DONE BY USER	WORK DONE BY TOOL
• VFB Design with Application and Ports • The Application Port Mapping, Signal Data Mapping and Service port mapping • Creation of Runnable, access point and Triggers • Map Runnable to OS Tasks	• Basic Data type for Interfaces will be available in tool. • Suggest the Port Mapping supported based on the interface. • Helps to create ports and interface in application level based on Signal Details configured in com stack. • Service port's interface will be pre-configured. • Helps in Error Correction, if there is any wrong/missing configuration. • Validate and Generate the Code

RUNNABLES AND TYPES

Runnable entity is a part of SWC where the application behavior logic is written. Runnable is analogous to functions in C

There are generally three types of runnables:

1. **Init Runnable:** This runnable is called on init of ECU
2. **Periodic Runnable:** This runnable is used when we need to trigger this runnable periodically to execute some operation periodically.
3. **Server Runnable:** This runnable is used to implement server of Client/Server port interface.

RTE EVENTS

All runnables should be mapped to RTE Events; else they will not be triggered . There are different RTE events

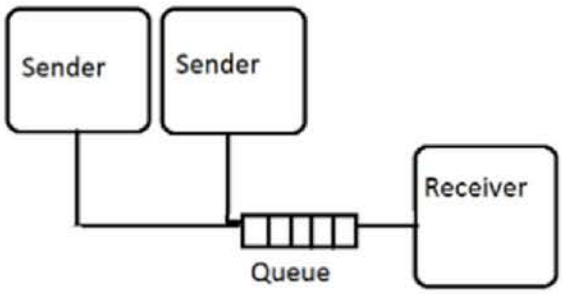
- **Timing Event:** Triggered at a set time/ periodically .
- **Data received event:** As the name suggests, such event will trigger a runnable whenever data is received by ports.
- **Operation Invoked Event:** This event is called by client when invoking a server runnable using Client/server port interface.

- **Mode Switch Event:** Whenever ECU mode is changed, runnables can be triggered to perform some work. For example ECU shutdown mode, if ECU needs to perform some work before shutdown, then such event shall be hooked with that runnable which will perform the work before shutdown.
- **Data Received Error Event:** whenever any error occurs in data receiving, a runnable can be called to take action on such event.
- **Data Send Completed Event:** This event will trigger a runnable if data is sent successfully to take action further on data transmission completion

SENDER RECEIVER – QUEUED VS NON QUEUED COMMUNICATION

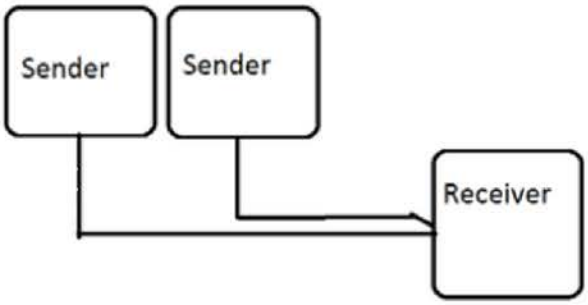
- SR is a communication pattern which offers asynchronous distribution of information where a sender communicates information to one or more receivers, or a receiver receives information from one or several senders.
- The process of sending data does not block the sender and the sender usually gets no response from the receivers.

QUEUED S/R COMMUNICATION



- If the sender-receiver communication is configured as queued, then the sent data is processed using a first-in-first-out (FIFO) queue with a specified length.
- Queued communication provides us a means to prevent the loss of data by storing and ordering the data received by one or multiple senders.

NON-QUEUED S/R COMMUNICATION



- If the sender-receiver communication is configured as non-queued, then the receiver always has access to the last sent data.

IMPLICIT VS EXPLICIT COMMUNICATION

ImplicitReceive	ExplicitReceive
• The input is always buffered. So, the input value remains the same in a single execution of the runnable, no matter how many times it is used. • This will be relevant when multiple ports are trying to write to a variable (Port) from different runnables • Implicit RTE Calls start with *RTE_IRead_* or RTE_IWrite_*.*	• The input is not buffered. So, the input value changes whenever it is used within a single execution of the runnable. • Explicit RTE calls start with *RTE_Read_* or RTE_Write_*.*

MULTIPLE INSTANCES

AUTOSAR provides a multi-instance concept for the Software Components. If a SWC is multi-instantiated; the instantiated components share the same code but separate memory (RAM) for the internal states.

SR SEQUENCE DIAGRAM

Refer figure 4.39, 4.40 in Autosar_SWS_RTE document

CS SEQUENCE DIAGRAM

Refer figure 4.43, 4.44 in Autosar_SWS_RTE document

DATA CONSISTENCY FEATURES

VARIABLES

Variables are created in the arxml file using Port Interface and Ports. They are used in Sender Receiver communication.. The application Software Component (ASWC) can update the status of the Variable using the rte_Write or other APIs that are generated based on the type of Sender Receiver Communication (explicit/Implicit/queued/non-queued). Another ASWC can read the updated variable from the RTE.c file using the Rte_Read API.

Inter-runnable Variables (IRV)

Inter-runnable Variables (IRV) are variables whose scope is limited to only one Software Component . The IRV can be accessed by different runnables inside the same software component. IRVs are created in the arxml file but they are not associated with ports or port interfaces as they are not shared between different Software Components. The IRV is created in the RTE.c file. The IRVs are accessed by the Software Components using the Rte_IrvRead and Rte_IrvWrite APIs.

Per-Instance Memory (Pim)

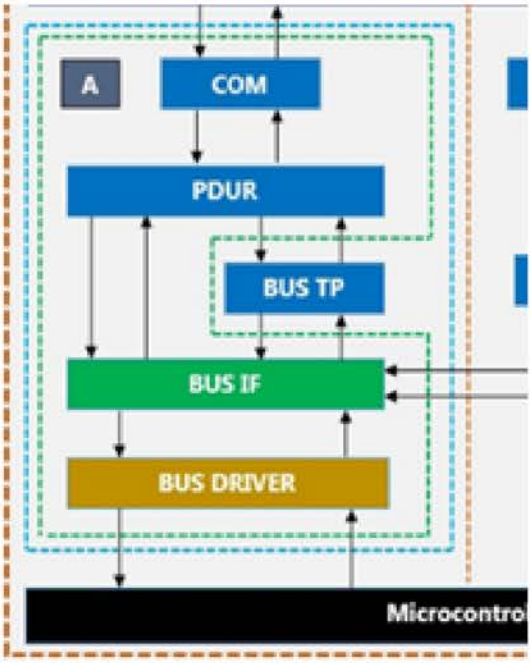
Per-Instance Memory (Pim) are also variables that get created in the RTE.c file. It is a global variable. The RTE creates a static memory for each software component. If we create a Pim of datatype uint8 and map it to a Software component, the RTE will allocate separate memory location for the Pim. If the software component is multiple instantiated then a separate memory location will be created for each instance. It is accessed by the application SWCs using the API *Rte_Pim_<n>; <n> stands for the instance of the software component.

Exclusive Areas (ExAr)

RTE shall support exclusive areas where a Runnable Entity or a Basic Software Schedulable Entity is declared as "running inside" the Exclusive Area. All Runnable Entities in a SW-Component that specify the same "runs inside" Exclusive Area shall be scheduled non pre-emptively.

COMMUNICATION STACK

DATA FLOW



COM — It has all the information about the Signals and Ipdu (can data without DLC , ID). It will take values from Application (ASWCs) layer through RTE and send to Pdur

PDUR — This layer has the details about the routing from where to where. Eg:- Com to Can Module or Com To Ethernet module etc....It is a static routing table configured by developer.

CAN TP/BUS TP — This Layer we will use if the Data is More that 8 Bytes. It will segment and reassemble data.

CAN IF/BUS IF — The module converts the PDU into CAN Frame. Identifier and DLC are added here to make it into CAN.

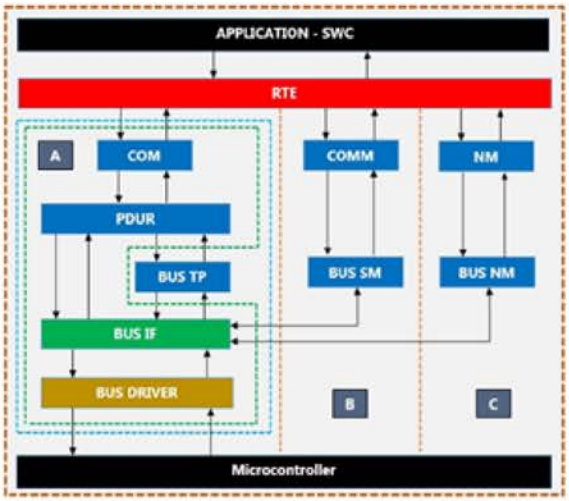
BUS DRIVERS — It has the drivers Api. The developer shall use the API to send data out of the bus or receive data from the bus

10 STEP TO DO AS A COM STACK DEVELOPER

1.	Load the EcuC and check if the signals in EcuC match to COM module(Tx and Rx)
2.	Mapping signals and Application Ports through RTE
3.	Ensuring Data flow in the right path
4.	Reconfiguring the PDUR if the signals are not aligned properly from top to bottom
5.	Add/Delete Signals/PDUs as defined in the ECU extract. Use an arxml viewer to make life easy
6.	Understand ECUC Pdus and how they are mapped between two SWCs
7.	Fix the errors
8.	Generate, Build and test
9.	Understand how to do PDU Routing in PDUR module
10.	Understand how to configure the Com Module
11.	Perform Signal Grouping ; Signal Gateway functionality if needed
12.	Understand how to write Call back functions

COMMUNICATION STACK

ROLE OF EACH COM STACK MODULE



- **Section A, B, C**
  - "A" for transmission/reception
  - "B" for Bus Status
  - "C" for Network Status

SECTION A

Section A is responsible only for the Transmission and reception of the Signals and Messages.

SECTION B

Section B will manage the status of the BUS

SECTION C

Section C will manage the Network

HOW TO READ THE COM STACK DOCUMENT

- Com Module Reference Doc :- *www.autosar.org/fileadmin/standards/classic/2211/AUTOSAR_SWS_COM.pdf*
- If you want the Sequence Diagram related Content you can refer 9.1 and 9.2 in the above link document
- If you want to read the Api definition you can refer 8.1 to 8.6 in the above link document

ECUC AND PDUS

ECUC It a ARXML file. It contains the information about the Data from top to Bottom.

PDUS

- I-pdu- It is in the services Layer.
- N-pdu- It is in the Network Layers
- L-Pdu- It is in the DataLink Layer

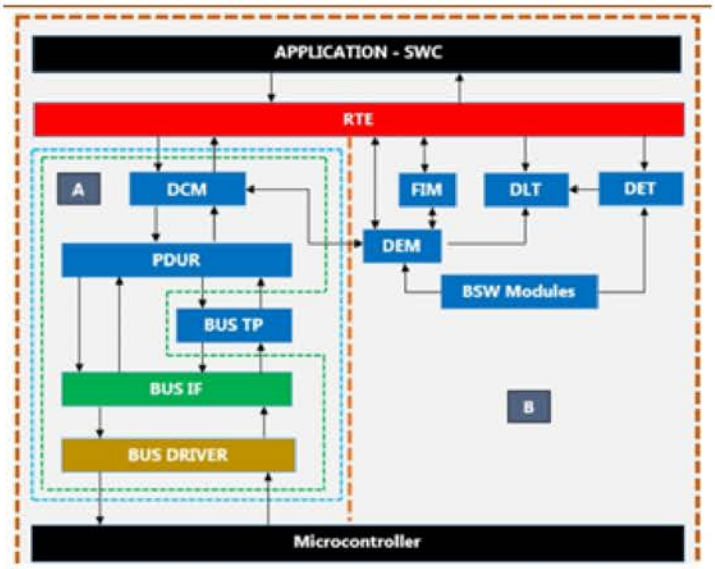
Ref:- www.linkedin.com/pulse/autosar-communication-service-cluster-carlos-enrique-hern%C3%A1ndez-ibarra

COMMON MISTAKES AND FIXES

- EcuC is not Mapped properly with corresponding modules.
- Data Mapping is not done properly with corresponding ports.
- Mismatch in configuration of transfer properties and Filters

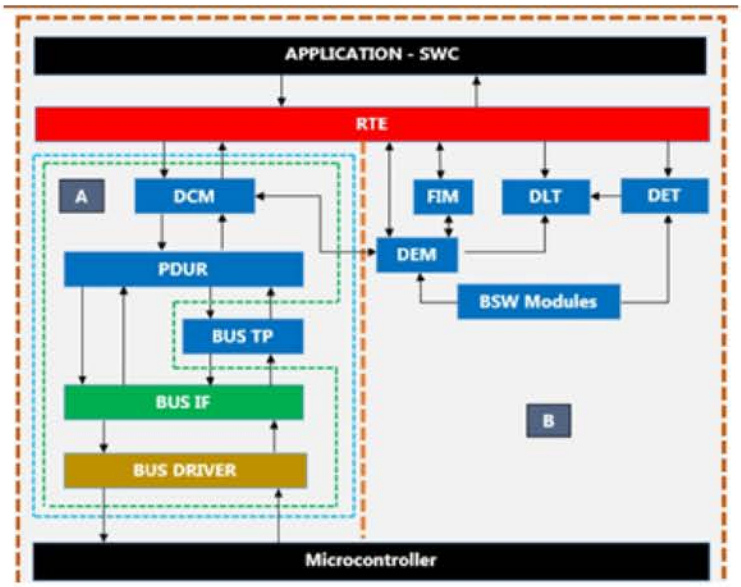
DIAGNOSTIC STACK

CAN VS CAN_DIAG SIGNAL SENDING MECHANISM



- In The Communication stack all the Pdus from the PDUR will routed to the Com Module
- In the DIag Stack all the Pdus from the PDUR will be routed to the DCM module

LOG ERROR USING DEM



In Application Layer, there are SWCs that are responsible for monitoring various events like Sensor voltage etc. When there is any discrepancy in the specified voltage then the Monitor SWC will use the look up table provided by the CDD (diagnostic data base file) to find the DTC code and log DTC(Diagnostic Trouble Code) in Non-Volatile Memory Location through DEM(Diagnostic Event manager) Module.

IMPLEMENTING DIAGNOSTIC SERVICE USING DCM

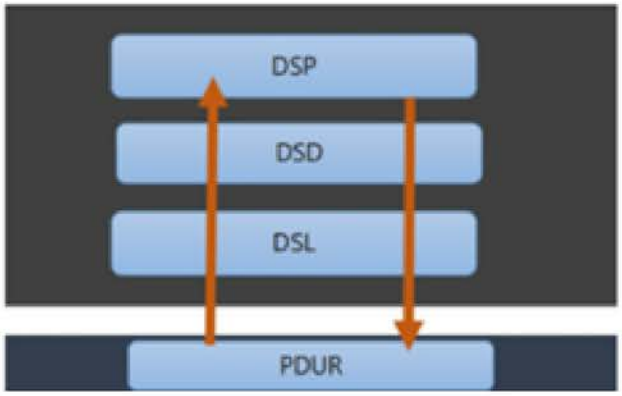
DCM is responsible for delivering all the dIagnostic services specified in the CDD file.

Eg1:- if Tester requests for DTC(Svc 0x19) , DCM will take the service and connect with DEM. DEM will take the value and provide to DCM which is returned to the TESTER.

Eg2:- If User is requesting for Routine Service(0x31), DCM will take the request and connect with Application Layer to service the request

DIAGNOSTIC STACK

DCM LAYERS



DCM has Three Layers

1 DSP(DIAGNOSTIC SERVICE PROCESSOR)

DSP is the Upper layer in DCM. It deals with the timing parameters and Processing the Diagnostic and Connecting with ASWCs(Application Layer).

2 DSD(DIAGNOSTIC SERVICE DISPATCHER)

DSD is the Middle layer in DCM. It deals with the Services that shall be served by DCM.

3 DSL(DIAGNOSTIC SERVICE LAYER)

DSL is the lowest layer in DCM. Its deals with the communication from DCM to PDUR and The Storing Buffer...

DIAGNOSTICS QUICK TIPS

- Check the Services and Sub Function are implemented in ECU or Not
- Check the Frame format is correct or Not.
- Check the dependencies are correct or not as per UDS Standard
- Check if the Diagnostic Console Matches or not.

MEMORY STACK

FLASH VS EEPROM

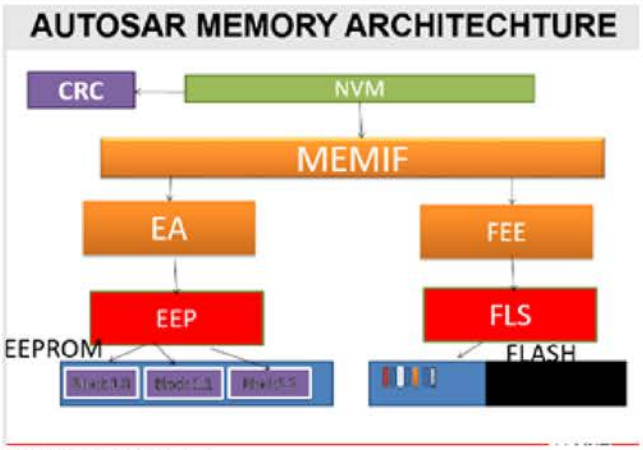
FLASH

- It has large memory element size usually one page. Individual pages cannot be erased. small blocks of data cannot be write/read
- It is time consuming

EEPROM

- Erasable and rewritable in small blocks(1-4 bytes)
- Data can be rewritten into the same memory cell rapidly
- Required External EEPROM which is costly.

MEMORY STACK FLOW



ROLE OF EACH MODULE

NVM	NVRAM Manager(NVM) is a service layer module that provides services of Memory to the Application(ASWCs), BSWM,DCM,DEM
MEMIF	It provides the abstraction from underlying modules FEE or EA Modules to Upper NVM. Eg:- NVM will request to Read or Write to MemIF module. MemIF will pass request to corresponding module.
EA	EEPROM Abstraction(Ea) Module facilitates abstraction from the addressing scheme of underlying EPPROM drivers
FEE	The Flash EEPROM Emulation (FEE) provides the upper layer with a Virtual addressing scheme, Segmentation as well as a "Virtually" Unlimited number of erase cycle.
EEP	EEPROM Driver(Eep) drivers provides services for reading and writing erasing from EEPROM
FLS	Flash Drivers(Fls) initialize flash and reads/write to flash memory.

All Variables that needs to be stored in the EEPROM have to saved as a NvMBlock

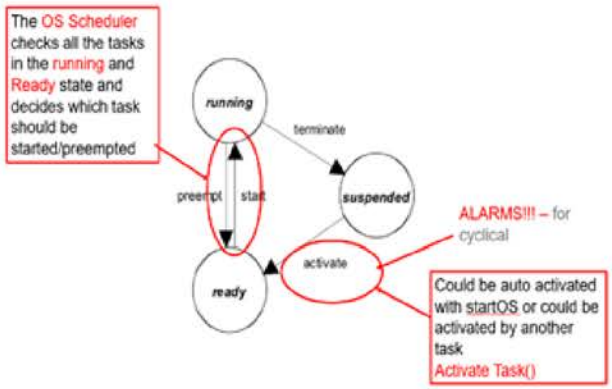
DIFFERENCE BETWEEN AUTOSAR OS AND OSEK OS

AUTOSAR OS — RTE is responsible for scheduling the runnables in both BSW and Application layer. OS supports the same

OSEK OS — OS is responsible for scheduling

BASIC TASK AND EXTENDED TASK

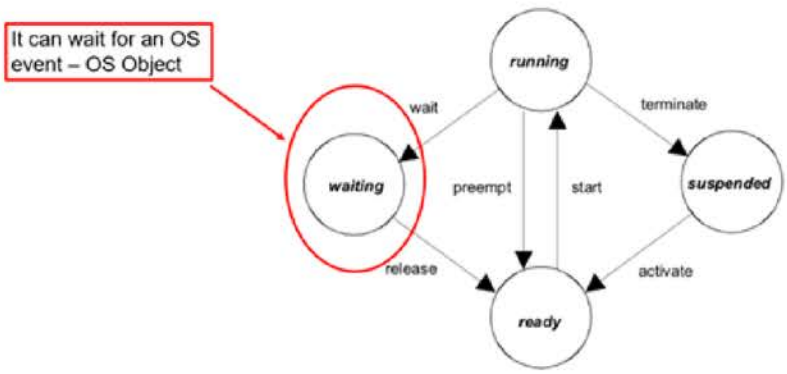
BASIC TASK



Basic Task has only three states - running , ready, suspended

- All runnables in default state will be in suspended state
- Rte triggers the OS task and the Task in turn allocates the resources for the runnables à ready state
- Runnables are executing à Running State
- Aysnchronous runnables / non blocking functions can be mapped to Basic Task

EXTENDED TASK



- It will have a Waiting State
- Synchronous runnables / blocking functions can be mapped to Extended Task
- Task will wait for server to return some value in Waiting State

PRE-EMPTIVE AND NON-PRE-EMPTIVE

Full pre-emptive scheduling will put the running task into the ready state, as higher priority task is called by RTE and serve the higher priority Task.

Non-Pre-Emptive - Higher priority task even if called by Rte will wait for the lower priority task to be served if the lower priority task is set as Non-Preemptive .

MAPPING RUNNABLE TO OS TASK

All Runnables are to be mapped to OS Task

ISR1 AND ISR2 INTERRUPTS

ISR1 - The ISR does not used any OS system service. After the ISR finished processing continue at the instruction where the interrupt has occurred.

ISR2 - The OSEK Operating System provides an ISR-frame to prepare a run-environment for a decided user routine.

STEPS OF EXECUTION

1)Create Runnables

2)Provide ports Access

3)Provide triggers/ RTE Events to Runnables

4)Create Task and map runnables to Task

5)Configure Task

1)Priority

2)Position

3)Task schedulingà Full or Non premprive

Task type

ANCIT

AUTOSAR SERVICES OFFERED BY ANCIT

- PoC Development in AUTOSAR standard
- Technical Consulting in development and Integration
- BSW stack Configuration and Integration
- Cdd Development and Integration
- Model based ASWC Development and integration
- Consulting - Architecture Design and Migration from Legacy to AUTOSAR
- Consulting - AUTOSAR Authoring tools development
- Hardware Design of Interface boards to integrate sensors/ actuators with the Microcontroller
- Training and Certification

FOR MORE DETAILS VISIT

www.ancitconsulting.com
www.ancitedutech/autosarservices
https://www.youtube.com/@ancittechnicalvideos

COIMBATORE
BANGALORE
DINDUGAL

beeshma@ancitconsulting.com
info@ancitconsulting.com

9840378602
9843541953