AUTOMOTIVE DOOR CONTROL SYSTEM STATIC DESIGN

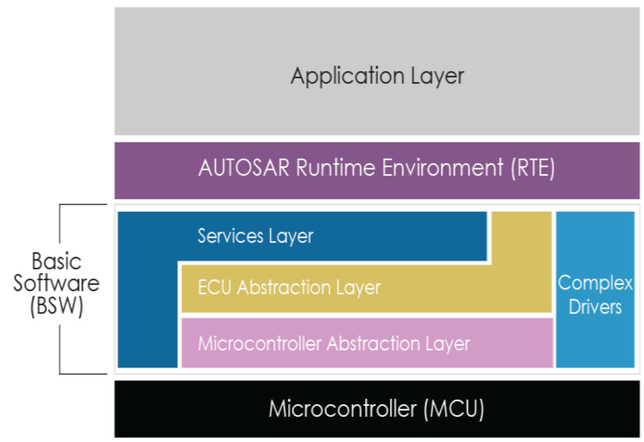
Nader Abd Elhalim Ahmed
Embedded Systems (Advanced Track)
March 30, 2023

AUTOSAR Architecture

Automotive Open System Architecture (AUTOSAR) is an open and standardized automotive software architecture, which supports standardization in interfaces between application software and basic vehicular functions and it helps in establishing common ECU software architecture for all the AUTOSAR members.

AUTOSAR is an open system architecture for automotive software development and provides standards for developing common automotive software applications. A growing and evolving standard defines a layered architecture for the software. The classic AUTOSAR platform runs on a microcontroller and is divided into 3 main layers; let us discuss them in detail:

- Basic Software Architecture- It is common to any AUTOSAR ECU.
- AUTOSAR Runtime Environment
- Application Layer



AUTOSAR Architecture

Basic Software Architecture (BSW)

AUTOSAR Basic Software Architecture consists of hundreds of software modules structured in different layers and is common to any AUTOSAR ECU. This means the supplier who has designed BSW can share it with other suppliers that are working on ECUs of engine, gearbox, etc.

❖ ***Basic software architecture in AUTOSAR consists of three layers:***

- **Microcontroller Abstraction Layer (MCAL):** MCAL is also known as a hardware abstraction layer and implements interface for the specific microcontroller. MCAL has layers of software, which are integrated with the microcontroller through registers, and offers drivers like system drivers, diagnostics drivers, memory drivers, communication drivers (CAN, LIN, Ethernet, etc.), I/O drivers and more.

- **ECU Abstraction Layer:** The prime task of the ECU abstraction layer is to deliver higher software layers ECU specific services. This layer and its drivers are independent of the microcontroller and dependent on the ECU hardware and provide access to all the peripherals and devices of ECU, which supports functionalities like communication, memory, I/O, etc.

- **Service Layer:** The service layer is the topmost layer of AUTOSAR Basic Software Architecture. The service layer constitutes an operating system, which runs from the application layer to the microcontroller at the bottom. The OS has an interface between the microcontroller and the application layer and can schedule application tasks. The service layer in BSW is responsible for services like network services, memory services, diagnostics service, communication service, ECU state management, and more.

❖ AUTOSAR Runtime Environment (RTE Layer)

AUTOSAR Run-time Environment is a middleware layer of the AUTOSAR software architecture between the BSW and the application layer and provides communication services for the application software.

❖ *Application Layer*

The application layer is the first layer of the AUTOSAR software architecture and supports custom functionalities implementation. This layer consists of the specific software components and many applications which perform specific tasks as per instructions.

After this short introduction, we will discuss **the Static design** of our project

THE REQUIRED COMPONENTS

1. Two microcontrollers (ECU1, ECU2)
2. One Door sensor (D)
3. One Light switch (L)
4. One Speed sensor (S)
5. Two lights, right (RL) and left (LL)
6. One buzzer (B)

COMMUNICATIONS BETWEEN COMPONENTS

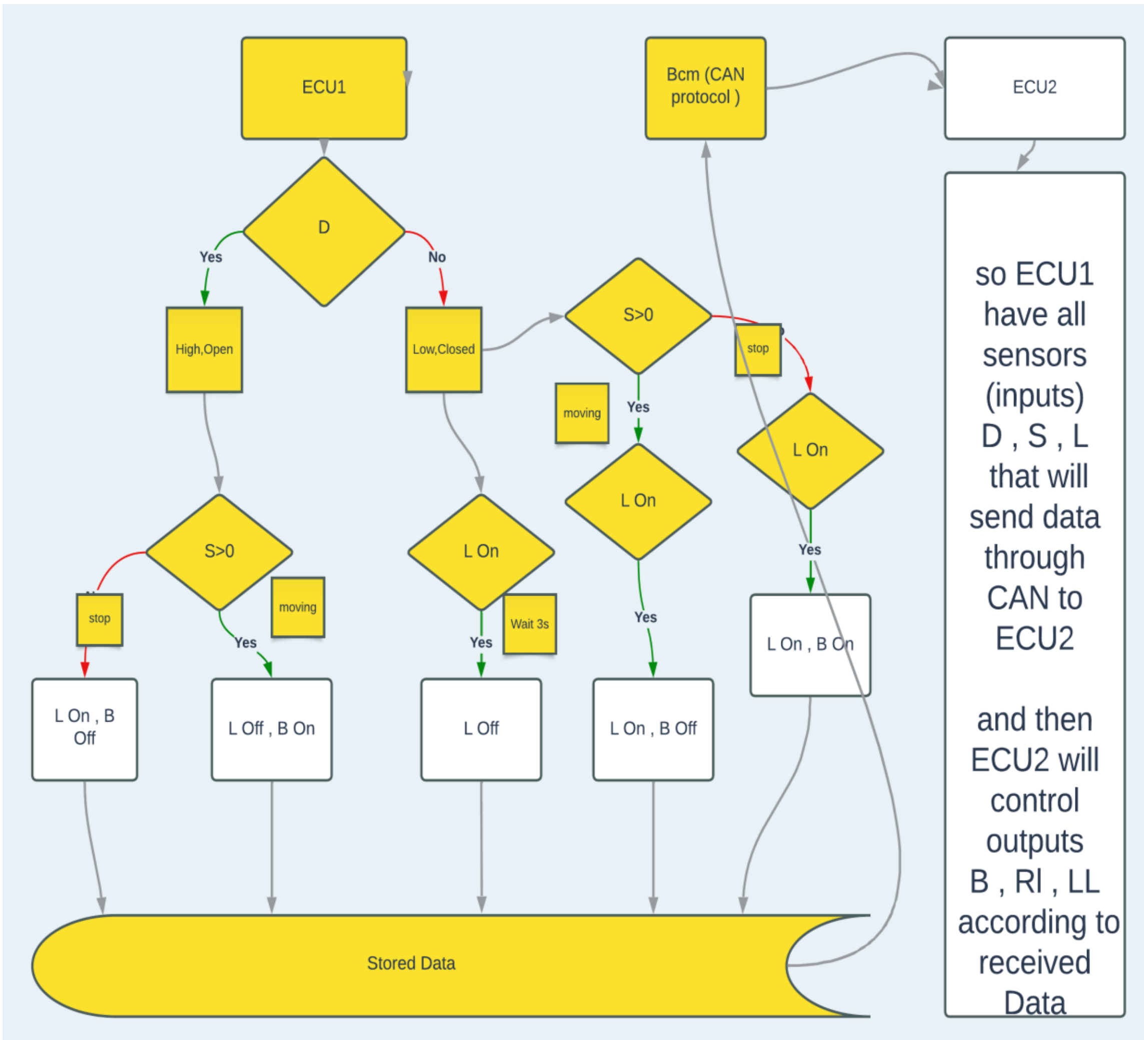
- Both of ECU1and ECU2 connected via CAN protocol
- All input devices (D, L, S) connected to ECU1
- All output devices (B, RL, LL) connected to ECU2

SOFTWARE REQUIREMENTS

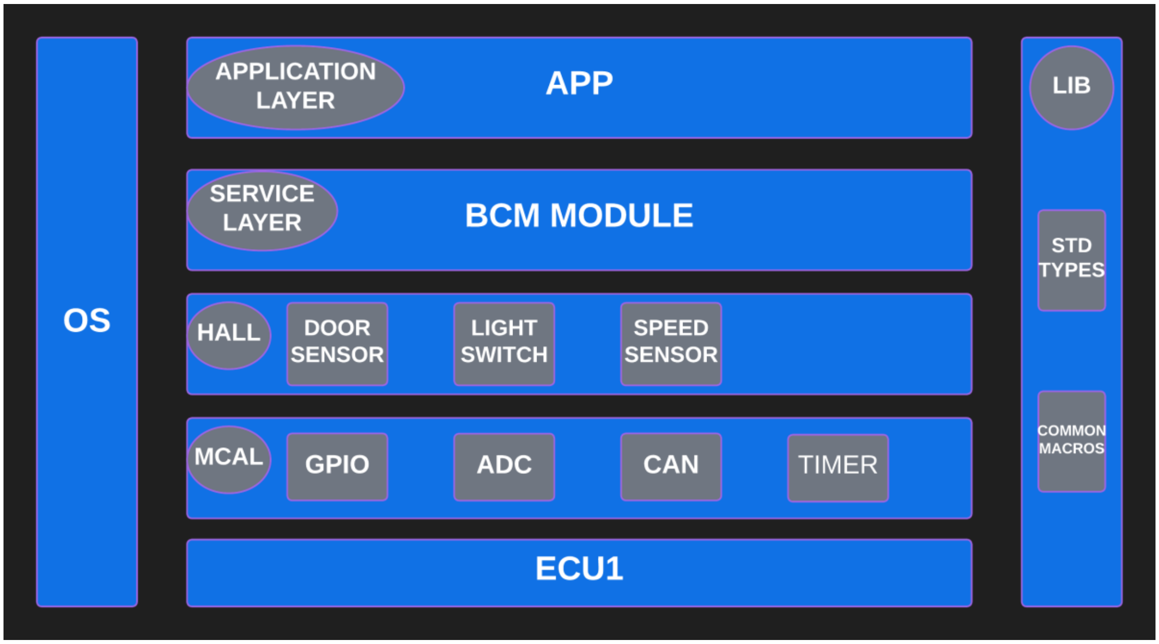
1. ECU 1 will send status messages periodically to ECU 2 through the CAN protocol
2. Status messages will be sent using Basic Communication Module (BCM)
3. Door state message will be sent every 10ms to ECU 2
4. Light switch state message will be sent every 20ms to ECU 2
5. Speed state message will be sent every 5ms to ECU 2
6. Each ECU will have an OS and application SW components
7. If the door is opened while the car is moving → Buzzer ON, Lights OFF
8. If the door is opened while the car is stopped → Buzzer OFF, Lights ON
9. If the door is closed while the lights were ON → Lights are OFF after 3 seconds
10. If the car is moving and the light switch is pressed → Buzzer OFF, Lights ON
11. If the car is stopped and the light switch is pressed → Buzzer ON, Lights ON

And this will be illustrated by Block Diagram in the following page

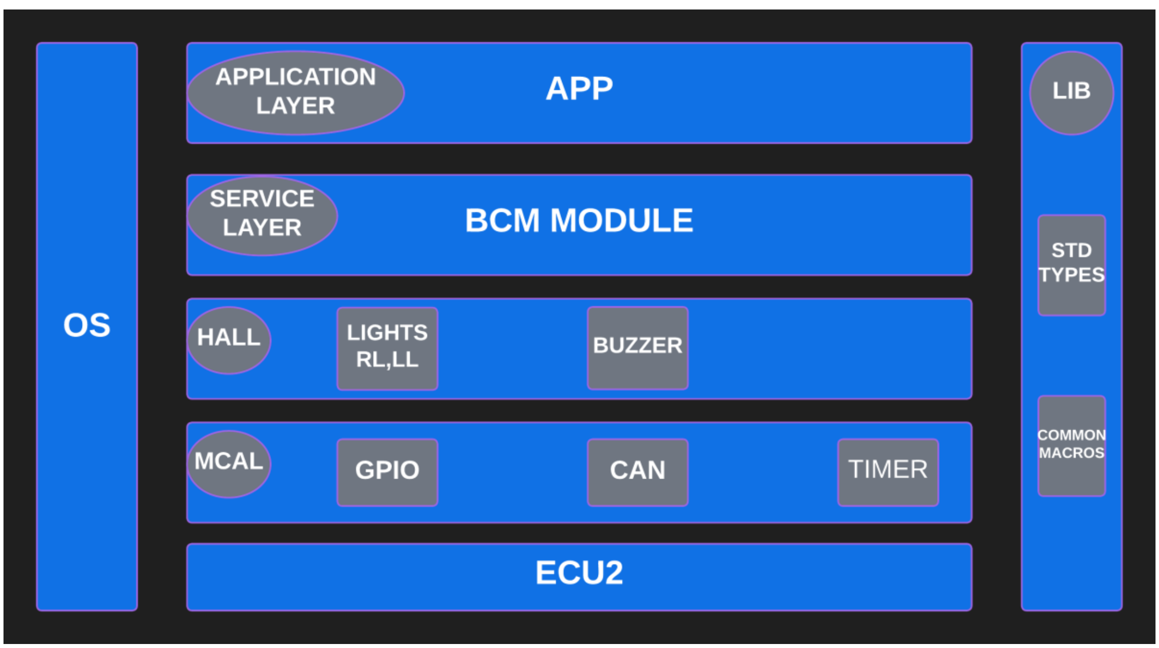
BLOCK DIAGRAM



ECU1 LAYERED ARCHITECTURE:



ECU2 LAYERED ARCHITECTURE:



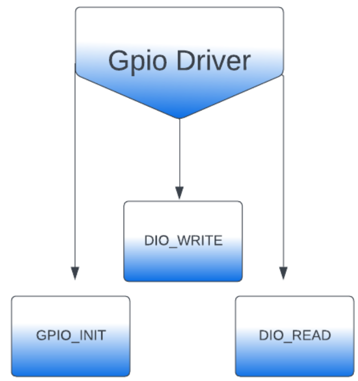
APIS FOR EACH MODULE & DESCRIPTION FOR THE USED TYPEDEFS :

- **COMMON MODULES/APIS IN ECU1&ECU2 :-**

#GPIO

TypeDefs

Name	TYPE	RANGE	Description
DIO_PinLevel	Enum	Low: 0 High : 1	
DIO_PinDir	Enum	Input: 0 Output : 1	
DIO_PortIDType	Enum	From(0 - 2)	Port A to Port C
DIO_PinIDType	Enum	From(0 - 7)	Pin 0 to Pin 7

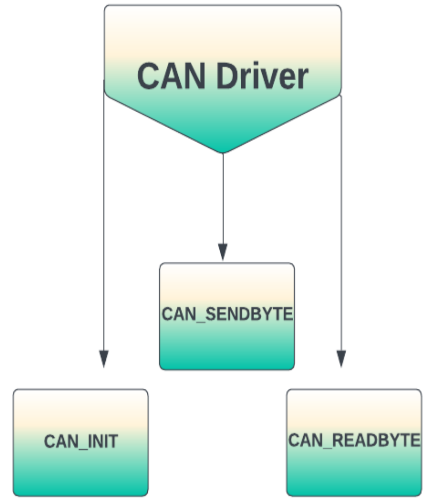


GPIO APIS

Name	Arguments	RANGE	Description
GPIO_INIT	GPIO_CfgPtr	Pointer to cfg parameter	GPIO_CfgPtr
DIO_WRITE	Pin_id Port_id Pin_Level	DIO_PinIDType DIO_PortIDType DIO_PinLevel	
DIO_READ	Pin_id Port_id	DIO_PinIDType DIO_PortIDType	Return pin level type (High :low)

#CAN

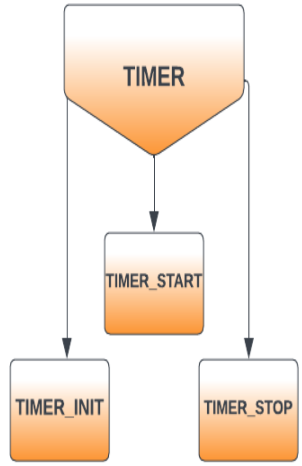
CAN APIS			
Name	**Arguments**	**RANGE**	**Description**
CAN_INIT	Can_ch		
CAN_SENDBYTE	- Can_ch type - *pointer to message	Ch1 or Ch2	Return 1 send ok 0 send failed
CAN_READBYTE	- Can_ch type	Ch1 or Ch2	Return Message that received -1 if receive failed



#TIMER

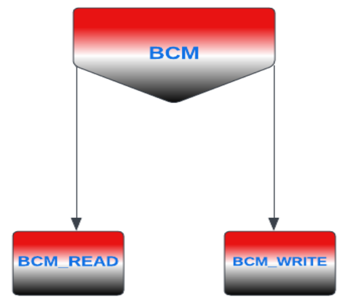
TypeDefs			
Name	**TYPE**	**RANGE**	**Description**
TIM_CH	Enum	TIMER0 0 TIMER1 1 TIMER2 2	
Tim_COUNT_DIR	Enum	up: 0 Down : 1	
TIM_sense	Enum	LEVEl: 0 Edge : 1	TIM_sense
TIM_LOAD	Uint32	From(0 - 2 ^31)	TIM_LOAD

TIMER APIS			
Name	**Arguments**	**RANGE**	**Description**
TIMER_Init	TIM_CH		
Timer_start	TIM_CH	TIMER0 TIMER1 TIMER2	Timer_start
Timer_stop	- TIM_CH	TIMER0 TIMER1 TIMER2	Timer_stop



#BCM

BCM APIS			
Name	**Arguments**	**RANGE**	**Description**
BCM_SENDBYTE	DATA	Uint8	Sends status message
BCM_READBYTE	N/A		Read message from CAN

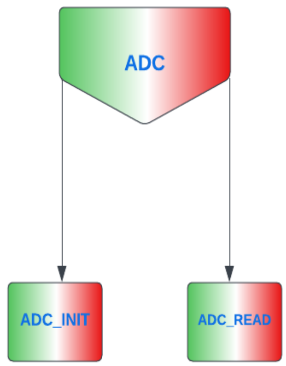


- **MODULES APIS IN ECU1:**

#ADC MODULE

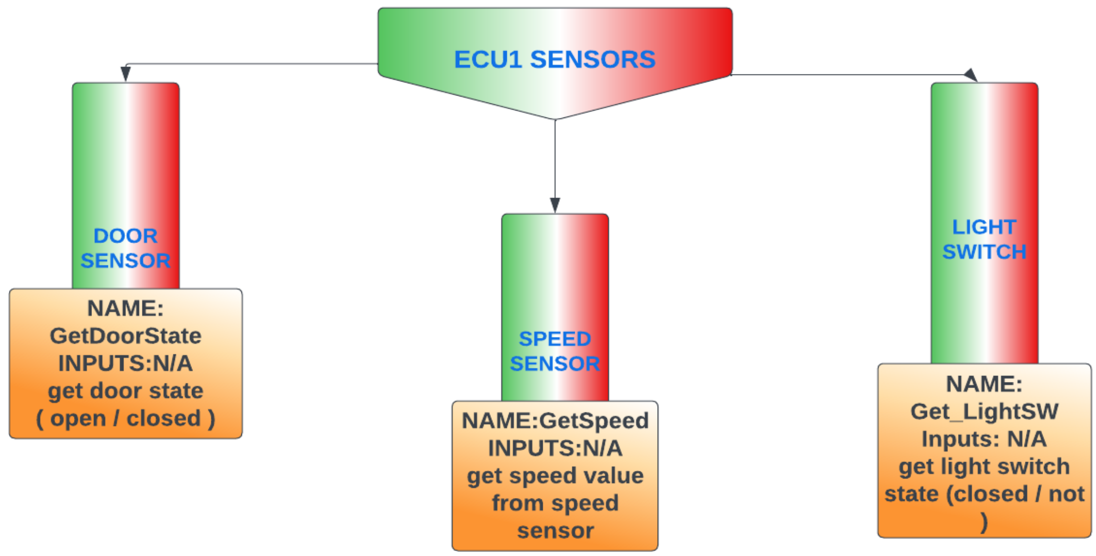
- **New data type:**

TypeDefs			
Name	**TYPE**	**RANGE**	**Description**
Channel_t	Enum	From: ADC0 to: ADC7	Representing ADC channels
Mode_t	Enum	Single Continuous	Representing ADC modes
ADC_Pre_t	Enum	ADC_2_Pre=1 ADC_4_Pre=2 ADC_8_Pre=3 ADC_64_Pre=6 ADC_128_Pre=7	Representing ADC prescalers
ADC_Volt_t	Enum	ADC_AREF= 0 ADC_AVCC= 1 ADC_INTERNA = 2	Pin 0 to Pin 7



ADC APIS			
Name	**Arguments**	**RANGE**	**Description**
ADC_ INIT	N/A	Uint8	Initialize ADC
ADC_READ	Channel Channel_t		Get value of selected ADC channel

#ECU1 APIS SENSORS:



#ECU2 APIS SENSORS: