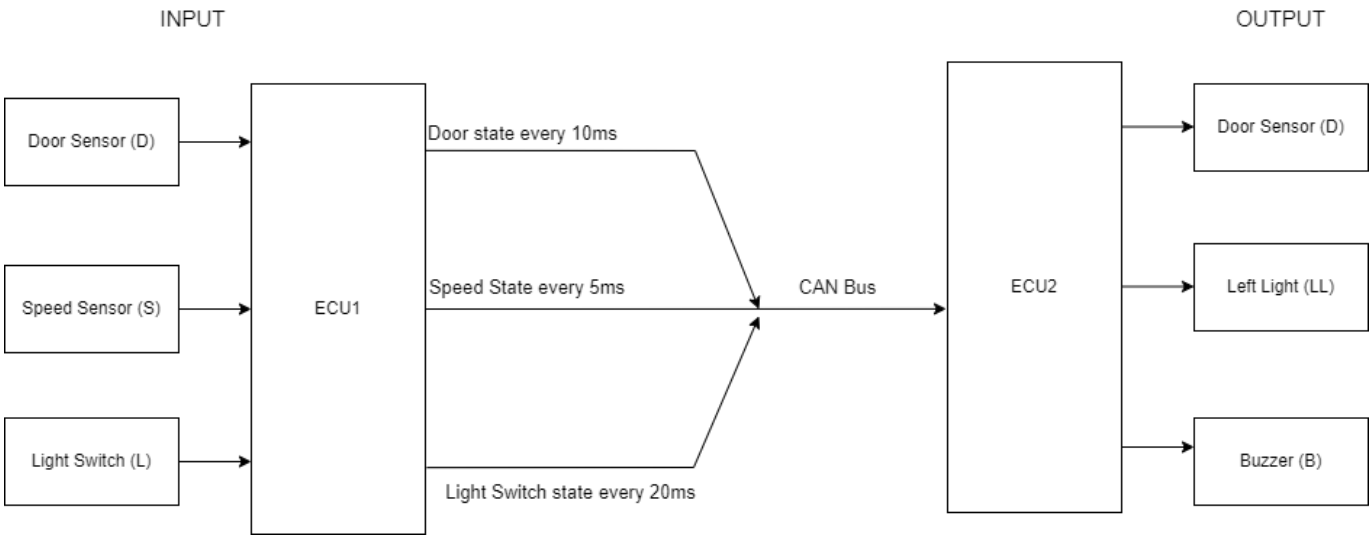
Static Design

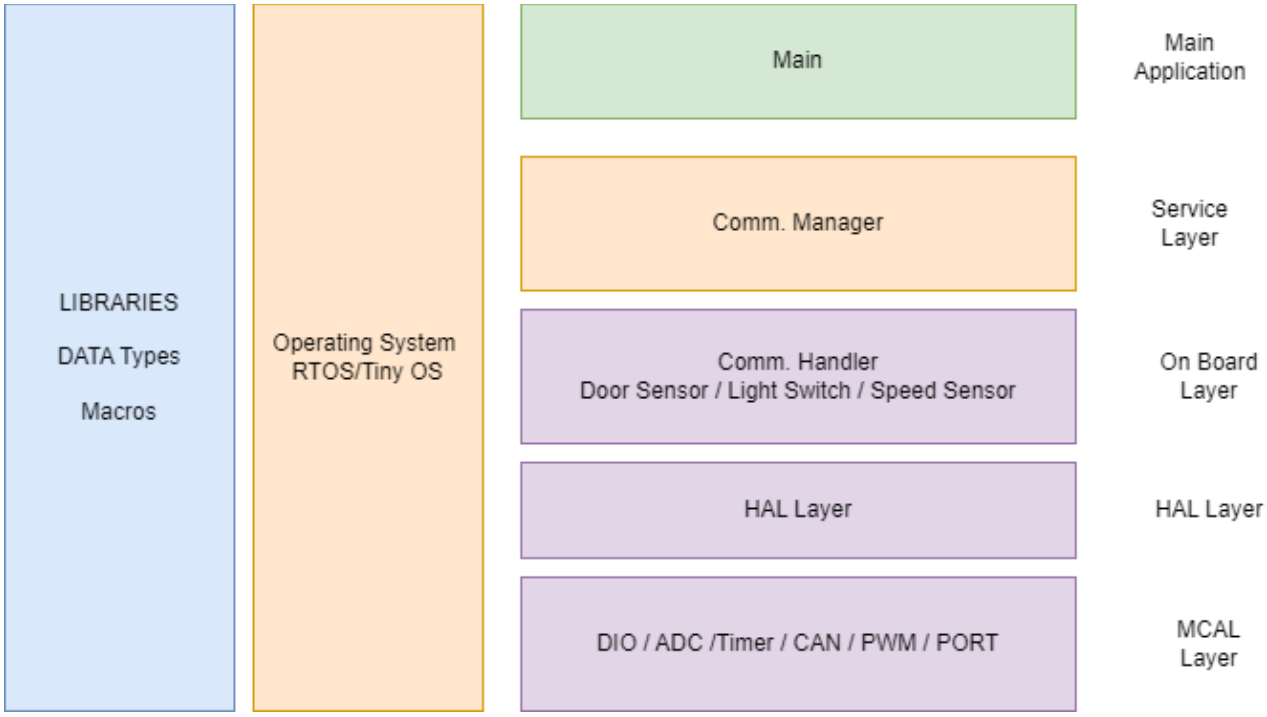
Name : Islam Mohamed Othman Mansour

Email : islam.othman2050@gmail.com

System Schematic



Layered Architecture for ECU 1



ECU Components and Modules:-

1-Components:-

1. CAN Bus
2. Light Switch
3. Door Sensor
4. Speed Sensor

2-MODULES:-

- CAN Transiever
- Door Sensor Module
- Light Switch Module
- Speed Sensor Module

3-Other Modules related to upper Lever Layers and it's Usage:-

- PORT Module --> Initialize all Pins Required for System to operate.
- ADC Module --> Used in Speed Sensor Module.
- CAN Module --> User for Sending and Receiving Data.
- Timer Module --> GPT for All modules either for orginising Tasks or triggering (Calling Tasks).
- DIO Module --> Used in Light Switch Module and in Door Sensor

3.1 API in Application Layer Description:-

Layer	Module	API		
Application	Main App.	Door Sensor Task	**Syntax:**	void DoorSensorTask(void);
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	**None**
			Return:	None
			Description:	Manage Door Sensor
Application	Main App.	Light Switch Task	**Syntax:**	void LightSwitchTask(void);
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	**None**
			Return:	None
			Description:	Manage Light Switch
Application	Main App.	SpeedSensor Task	**Syntax:**	void SpeedSensorTask(void);
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	**None**
			Return:	None
			Description:	Manage Speed Sensor

3.2 API in Serviec Layer Description:-

Layer	Module	API		
Service Layer	Comm. Manager	CM_Manager	**Syntax:**	void CM_Manager (uint8 BUS_ID, uint64 Data);
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	**BUS_ID**: that the ID commutation protocol want to connect it, **Data** : Data to be sent over desired -selected- Bus.
			Return:	None
			Description:	Manage CAN Transmitter by selecting CAN Bus ID
Service Layer	Comm. Manager	Sensor Manager	**Syntax:**	State_Level Sensor_Manager(sensor_ID Sensor_ID);
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	Sensor_ID: ID of the desired sensor to be read.
			Return:	Data read from sensor –State-
			Description:	Manage request of reading data from desired sensor.

Definition of Some Data-Types :

Data Type	Description
typedef unsigned char uint8	Used in defining ID for Busses selection and it varies from 0 to 255 IDs. And it is stored in 1Byte for every used_bus.
typedef unsigned long long uint64	Because of maximum capacity for CAN frame is 64bit, it is here used to fill these frames.
Typedef enum {LOW,HIGH} State_Lavel	Used to define different states of sensor. And it ranges between LOW and HIGH or even more and takes 1 bit from memory. Low = 0 High = 1

Typedef enum {Sensor_1,...} Sensor_ID	Used to define different no. of sensors. And it ranges between Sensor_1 until N_of_Sensors. Size : 1 bit for every sensor. We have only 3 sensors so our range is between 0 and 2. Sensor_1 = 0 Sensor_2 = 1 Sensor_N = N-1

3.3 API in On-Board Layer:-

Layer	Module	API		
On-Board Layer	Comm. Handler	CM_Handler	**Syntax:**	void CM_Handler (uint8 BUS_ID, uint64 Data);
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	**BUS_ID**: that the ID commutation protocol want to connect it, **Data** : Data to be sent over desired -selected- Bus By CM_Manager.
			Return:	None
			Description:	Handle request of CAN Transmitter by selecting CAN Bus ID
On-Board Layer	Comm. Handler	Sensor Handler	**Syntax:**	State_Level Sensor_Manager(sensor_ID Sensor_ID);
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	Sensor_ID: ID of the desired sensor to be read.
			Return:	Data read from sensor –State-
			Description:	Manage request of reading data from desired sensor but communicate with HW direct.

On-Board Layer	Door Sensor	Door Sensor_Init	**Syntax:**	Void Door_SensorInit(void);
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	none
			Return:	none
			Description:	Initialize Door Sensor PINS and Modes if they use DIO or ADC or I2C.
		Door Sensor_Read	**Syntax:**	DoorState Door_SensorRead(void);
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	none
			Return:	DoorState
			Description:	Get reading from door sensor if it is open or closed.
On-Board Layer	Light Switch	LightSwitch_Init	**Syntax:**	Void LightSwitch_init(void)
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	none
			Return:	none
			Description:	Initilize light switch Pins.
		LightSwitch_Read	**Syntax:**	LightState LightSwitch_Read(void)
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	none
			Return:	LightState if it is open or closed.
			Description:	Read light switch state if it's pressed(open) or not.
On-Board Layer	Speed Sensor	SpeedSensor_init	**Syntax:**	Void SpeedSensor_init(void)
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	none
			Return:	none
			Description:	Initilize Speed Sensor Pin in ADC mode.
		SpeedSensor_Read	**Syntax:**	MovementState SpeedSensor_Read(void)
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	none
			Return:	MovementState if the

					car is moving or not
				Description:	Read Speed Sensor and get the status of movement back. (Moving or Stop)

Definition of Some Data-Types :

Data Type	Description
DoorState	Typedef enum {closed , open}DoorState; Closed = 0 Open = 1 Size -> 1bit
LightState	Typedef enum {not_press, press}LightState; Not_press = 0 Press = 1 Size -> 1bit
MovementState	Typedef enum {stop, moving}MovementState; Stop = 0 Moving = 1

	Size -> 1bit

3.4 API in MCAL Layer Description:-

<table>
<tr><th>Layer</th><th>Module</th><th>API</th><th></th></tr>
<tr><td>MCAL Layer</td><td>DIO</td><td>DIO_Init</td><td>
<table>
<tr><td>Syntax:</td><td>Void DIO_Init(void)</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>none</td></tr>
<tr><td>Return:</td><td>None</td></tr>
<tr><td>Description:</td><td>Initialize DIO Pins in Cfg file with predefined Configurations.</td></tr>
</table>
</td></tr>
<tr><td></td><td></td><td>DIO_Write</td><td>
<table>
<tr><td>Syntax:</td><td>Void DIO_Write(DIO_ID Pin,DIO_State Level)</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>DIO_ID -> Id of the desired Pin
DIO_State -> desired Level to be written (HIGH, LOW)</td></tr>
<tr><td>Return:</td><td>None</td></tr>
<tr><td>Description:</td><td>Write Level on PIN Channel and it ranges between high and LOW.</td></tr>
</table>
</td></tr>
<tr><td></td><td></td><td>DIO_Read</td><td>
<table>
<tr><td>Syntax:</td><td>DIO_State DIO_Read(DIO_ID Pin)</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>DIO_ID -> Id of the desired Pin to be read.</td></tr>
<tr><td>Return:</td><td>DIO_State -> Level to be read on desired Pin(HIGH, LOW)</td></tr>
<tr><td>Description:</td><td>Read status of DIO pin.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr>
<td>MCAL Layer</td>
<td>PORT</td>
<td>Port_init(*PORT_Cfg)</td>
<td>
<table>
<tr><td>Syntax:</td><td>Void PORT_Init(*PORT_Cfg)</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>Takes a pointer to preconfigured array of PORTs to Initialize it’s Pins as configured in Cfg files.</td></tr>
<tr><td>Return:</td><td>none</td></tr>
<tr><td>Description:</td><td>Initialize each written Port in array with desired Configs.</td></tr>
</table>
</td>
</tr>
<tr>
<td>MCAL Layer</td>
<td>TIMER</td>
<td>Timer Init

Timer Start

Timer Stop</td>
<td>
<table>
<tr><td>Syntax:</td><td>Void Timer_init(void);</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>none</td></tr>
<tr><td>Return:</td><td>none</td></tr>
<tr><td>Description:</td><td>Initalize and Configure Timer</td></tr>
</table>

<table>
<tr><td>Syntax:</td><td>Void Timer_Start(Timer_Channel Channel , Timer_Val Value);</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>Timer_Channed -> Channel that we want to connect it to the Timer.
Timer_Val-> Value at which the timer will react it can be overflow value or compare match value depends on configs.</td></tr>
<tr><td>Return:</td><td>none</td></tr>
<tr><td>Description:</td><td>Start the Timer and Connect it to desired channel + Configure max. Value to be reached.</td></tr>
</table>

<table>
<tr><td>Syntax:</td><td>Void Timer_Stop(Timer_Channel Channel);</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>Timer_Channel -> Channel to be cut from timer.</td></tr>
<tr><td>Return:</td><td>none</td></tr>
<tr><td>Description:</td><td>Stop the Timer and disconnect the timer from desired channel</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>MCAL Layer</td>
<td>CAN</td>
<td>CAN Init

CAN Trasnmitt</td>
<td>
<table>
<tr><td>Syntax:</td><td>Void CAN_init(void)</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>none</td></tr>
<tr><td>Return:</td><td>none</td></tr>
<tr><td>Description:</td><td>Initialize CAN Bus and Configure it.</td></tr>
</table>

<table>
<tr><td>Syntax:</td><td>Void CAN_Transmitt(uint8 Channel , uint64 Data)</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>none</td></tr>
<tr><td>Return:</td><td>Uint8 Channel-> Pin on which data will be transmitted.
uint64 Data -> Data to be sent</td></tr>
<tr><td>Description:</td><td>Transmitt desired Data over Desired Channel.</td></tr>
</table>
</td>
</tr>
<tr>
<td>On-Board Layer</td>
<td>ADC</td>
<td>ADC_Init

ADC_Read</td>
<td>
<table>
<tr><td>Syntax:</td><td>Void ADC_init(void)</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>none</td></tr>
<tr><td>Return:</td><td>none</td></tr>
<tr><td>Description:</td><td>Initialize and Configure ADC.</td></tr>
</table>

<table>
<tr><td>Syntax:</td><td>Uint16 ADC_Read(DIO_ID Pin)</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>DIO_ID Pin -> Takes Desired Pin to be read.</td></tr>
<tr><td>Return:</td><td>Reading from desired pin and it varies from 0 to 65535.</td></tr>
<tr><td>Description:</td><td>Read desired Pin in ADC mode with a resolution of 2^16.</td></tr>
</table>
</td>
</tr>
</table>

Definition of Some Data-Types :

Data Type	Description

DIO_ID	Typedef {Channel_1,…..,Channel_N}DIO_ID; Channel_1 = 0 Channel_2 = 1 Channel_N = N-1 Size = N bytes.
DIO_State	Typedef enum {LOW ,HIGH }DIO_State; LOW = 0 HIGH = 1 Size -> 1bit
Timer_Channel	Typedef enum {Timer_1, Timer_2}Timer_Channel; Size -> N bytes depends on How many Timers do we Have.
Timer_Val	Typedef uint32 Timer_Val; Range of Timer_Ticks varies from 0 -> 2^32-1.

1. folder structure according to the previous points:

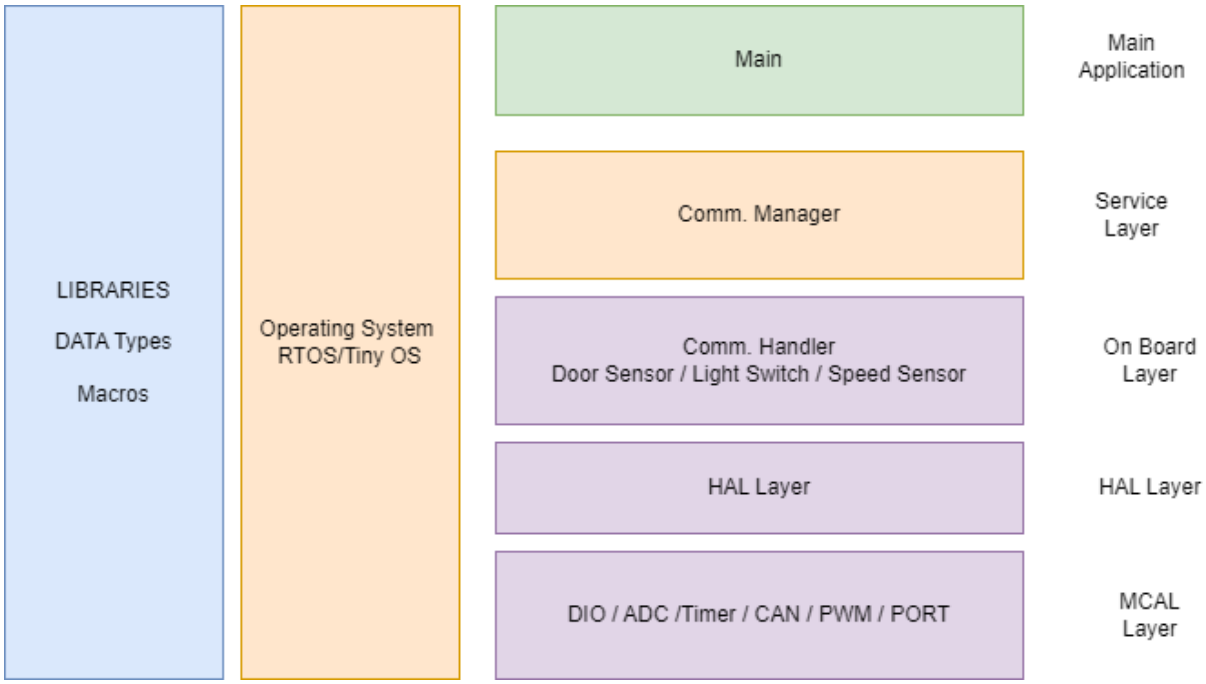
Application folder	Servies folder	On Board Layer
main.c	Operting_system.c	CM_Handler.c

	CM_Manager.c	Sensor_Handler.c
	Sensor_Manager.c	Door_sensor.c
		Light_switch.c
		Speed_sensor.c

MCAL folder	Configure folder
dio.c	Timer_config.c
port.c	Adc_config.c
adc.c	Can_config.c
Timer.c	Port_config.c
can.c	Dio_config.c
	Door_sensorconfig.c
	Light_switchconfig.c
	Speed_sensorconfig.c

Commen folder (all the header (name.h))
Mainapp.h / os.h / servies.h
Comm_manager.h/Sonser_manager.h
Light_switch.h / speed_sonser.h / Door_sensor.h
Dio.h / port.h / timer.h /can.h/adc.h
dio_config.h/port_config.h / timer_config.h /can_config.h /adc_config.h
Stdtypes.h /common_macro.h /Hw.h

Layered Architecture for ECU 2



ECU Components and Modules:-

1-Components:-

1. CAN Bus
2. Light Left
3. Light Right
4. Buzzer

2-MODULES:-

- CAN Transiever
- Light Left Module
- Light Right Module
- Buzzer Module

--Internal

- PORT Module
- DIO Module
- CAN Module

- Timer Module

3.1 API in Application Layer:-

Layer	Module	API		
Application	main	Periodic_Receive	**Syntax:**	void Periodic_Receive (uint64* Data, uint8* Bus_ID);
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	**Pointer to Buffer and Pointer to Bus to be monitored**
			Return:	None
			Description:	Manage Data received periodically from ECU_1

3.2 API in Service Layer:-

Layer	Module	API		
Service Layer	Comm. Manage	CM_Manager	**Syntax:**	Uint64 CM_Manager (uint8 Bus_ID);
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	**BUS_ID at which we want to receive data.**
			Return:	Received data from CAN Bus with 64bit width from ECU_1.
			Description:	Manage request data received over CAN Bus from ECU_1
Service Layer	Comm. Manager	Device Manager	**Syntax:**	void Device_Manager(Device_ID Device Device_Action Action);
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	Device_ID: ID of the desired Device (Buzzer – LL - Rl)and desired action to do with it.
			Return:	none
			Description:	Manage request of action to do with desired Device.

Types define of argument of APIs:

Data Type	Description
Bus_ID	Typedef unsigned char BUS_ID Used to handle ID for CAN Busses and can carry upto 256 Bus. Size = 8 bytes.
Device_ID	Typedef enum {Device_1, ……,Device_N}Device_ID; Size -> Nbit in our case 3 bits.
Device_Action	Typedef enum {OFF, ON}Device_Action; Used to handle different actions of Devices. Size -> 2bits

3.3 On_Board in Service Layer:-

Layer	Module	API	Field	Value
On-Board Layer	Comm. Handler	CM_Handler	**Syntax:**	Uint64 CM_Manager (uint8 Bus_ID);
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	**BUS_ID at which we want to receive data.**
			Return:	Received data from CAN Bus with 64bit width from ECU_1.
			Description:	Handle request data received over CAN Bus from ECU_1
On-Board Layer	Comm. Handler	Device Manager	**Syntax:**	void Device_Handler(Device_ID Device Device_Action Action);
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	Device_ID: ID of the desired Device (Buzzer – LL - Rl)and desired action to do with it.
			Return:	none
			Description:	Handle request of action to do with desired Device.
On-Board Layer	Buzzer	Buzzer Init	**Syntax:**	void Buzzer_Init(void);
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	none
			Return:	none
			Description:	Initialize Buzzer pin as pre configured
		Buzzer On	**Syntax:**	void Buzzer_ON(void);
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	none
			Return:	none
			Description:	Turns Buzzer On
			Syntax:	void Buzzer_OFF(void);
			Sync/Async:	Synchronous
			Reentrancy:	Non-Reentrant
			Parameters:	none
			Return:	none
			Description:	Turns Buzzer Off

<table>
<tr><td></td><td></td><td>Buzzer Off</td><td></td></tr>
<tr><td>On-Board Layer</td><td>Light</td><td>Light Init</td><td>
<table>
<tr><td>Syntax:</td><td>void Light_Init(void);</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>none</td></tr>
<tr><td>Return:</td><td>none</td></tr>
<tr><td>Description:</td><td>Initialize preconfigured Pin for Left and Right Lights.</td></tr>
</table>
</td></tr>
<tr><td></td><td></td><td>Light On</td><td>
<table>
<tr><td>Syntax:</td><td>void Light-On(Light_ID Light);</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>Light_ID -> ID which identifies the Light is Left or Right.</td></tr>
<tr><td>Return:</td><td>none</td></tr>
<tr><td>Description:</td><td>Turns desired Light On</td></tr>
</table>
</td></tr>
<tr><td></td><td></td><td>Light Off</td><td>
<table>
<tr><td>Syntax:</td><td>void Buzzer_OFF(Light_ID Light);</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>Light_ID -> ID which identifies the Light is Left or Right.</td></tr>
<tr><td>Return:</td><td>none</td></tr>
<tr><td>Description:</td><td>Turns desired Light Off</td></tr>
</table>
</td></tr>
</table>

Types define of argument of APIs:

Data Type	Description
Light_ID	Typedef enum {Left_Light , Right_Light}Light_ID; Left_Light = 0 Right_Light = 1 This typedef is used to differ between left and right light. Size = 2 bits.

3.4 API in MCAL Layer Description:-

Layer	Module	API	
MCAL Layer	DIO	DIO_Init	**Syntax:** Void DIO_Init(void) **Sync/Async:** Synchronous **Reentrancy:** Non-Reentrant **Parameters:** none **Return:** None **Description:** Initialize DIO Pins in Cfg file with predefined Configurations.
		DIO_Write	**Syntax:** Void DIO_Write(DIO_ID Pin,DIO_State Level) **Sync/Async:** Synchronous **Reentrancy:** Non-Reentrant **Parameters:** DIO_ID -> Id of the desired Pin DIO_State -> desired Level to be written (HIGH, LOW) **Return:** None **Description:** Write Level on PIN Channel and it ranges between high and LOW.
		DIO_Read	

<table>
<tr><td></td><td></td><td></td><td>
<table>
<tr><td>Syntax:</td><td>DIO_State DIO_Read(DIO_ID Pin)</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>DIO_ID -> Id of the desired Pin to be read.</td></tr>
<tr><td>Return:</td><td>DIO_State -> Level to be read on desired Pin(HIGH, LOW)</td></tr>
<tr><td>Description:</td><td>Read status of DIO pin.</td></tr>
</table>
</td></tr>
<tr><td>MCAL Layer</td><td>PORT</td><td>Port_init(*PORT_Cfg)</td><td>
<table>
<tr><td>Syntax:</td><td>Void PORT_Init(*PORT_Cfg)</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>Takes a pointer to preconfigured array of PORTs to Initialize it’s Pins as configured in Cfg files.</td></tr>
<tr><td>Return:</td><td>none</td></tr>
<tr><td>Description:</td><td>Initialize each written Port in array with desired Configs.</td></tr>
</table>
</td></tr>
<tr><td>MCAL Layer</td><td>TIMER</td><td>Timer Init

Timer Start</td><td>
<table>
<tr><td>Syntax:</td><td>Void Timer_init(void);</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>none</td></tr>
<tr><td>Return:</td><td>none</td></tr>
<tr><td>Description:</td><td>Initalize and Configure Timer</td></tr>
</table>

<table>
<tr><td>Syntax:</td><td>Void Timer_Start(Timer_Channel Channel , Timer_Val Value);</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>Timer_Channed -> Channel that we want to connect it to the Timer.
Timer_Val-> Value at which the timer will react it can be overflow value or compare match value depends on configs.</td></tr>
<tr><td>Return:</td><td>none</td></tr>
<tr><td>Description:</td><td>Start the Timer and Connect it to desired channel + Configure max. Value to be reached.</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td></td><td>Timer Stop</td><td>
<table>
<tr><td>Syntax:</td><td>Void Timer_Stop(Timer_Channel Channel);</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>Timer_Channel -> Channel to be cut from timer.</td></tr>
<tr><td>Return:</td><td>none</td></tr>
<tr><td>Description:</td><td>Stop the Timer and disconnect the timer from desired channel</td></tr>
</table>
</td></tr>
<tr><td rowspan="2">MCAL Layer</td><td rowspan="2">CAN</td><td>CAN Init</td><td>
<table>
<tr><td>Syntax:</td><td>Void CAN_init(void)</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>none</td></tr>
<tr><td>Return:</td><td>none</td></tr>
<tr><td>Description:</td><td>Initialize CAN Bus and Configure it.</td></tr>
</table>
</td></tr>
<tr><td>CAN Trasnmitt</td><td>
<table>
<tr><td>Syntax:</td><td>Void CAN_Transmitt(uint8 Channel , uint64 Data)</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>none</td></tr>
<tr><td>Return:</td><td>Uint8 Channel-> Pin on which data will be transmitted.
uint64 Data -> Data to be sent</td></tr>
<tr><td>Description:</td><td>Transmitt desired Data over Desired Channel.</td></tr>
</table>
</td></tr>
<tr><td rowspan="2">On-Board Layer</td><td rowspan="2">ADC</td><td>ADC_Init</td><td>
<table>
<tr><td>Syntax:</td><td>Void ADC_init(void)</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>none</td></tr>
<tr><td>Return:</td><td>none</td></tr>
<tr><td>Description:</td><td>Initialize and Configure ADC.</td></tr>
</table>
</td></tr>
<tr><td>ADC_Read</td><td>
<table>
<tr><td>Syntax:</td><td>Uint16 ADC_Read(DIO_ID Pin)</td></tr>
<tr><td>Sync/Async:</td><td>Synchronous</td></tr>
<tr><td>Reentrancy:</td><td>Non-Reentrant</td></tr>
<tr><td>Parameters:</td><td>DIO_ID Pin -> Takes Desired Pin to be read.</td></tr>
</table>
</td></tr>
</table>

			Return:	Reading from desired pin and it varies from 0 to 65535.
			Description:	Read desired Pin in ADC mode with a resolution of 2^16.

Definition of Some Data-Types :

Data Type	Description
DIO_ID	Typedef {Channel_1,…..,Channel_N}DIO_ID; Size = N bytes.
DIO_State	Typedef enum {LOW ,HIGH }DIO_State; LOW = 0 HIGH =1 Size -> 1bit
Timer_Channel	Typedef enum {Timer_1, Timer_2}Timer_Channel; Size -> N bytes depends on How many Timers do we Have.
Timer_Val	Typedef uint32 Timer_Val; Range of Timer_Ticks varies from 0 -> 2^32-1.

2. folder structure according to the previous points:

Application folder	Servies folder	On Board Layer
main.c	Operting_system.c	CM_Handler.c
	CM_Manager.c	Device_Handler.c
	Sensor_Manager.c	Buzzer.c
		Light.c

MCAL folder	Configure folder
dio.c	Timer_config.c
Timer.c	Port_config.c
can.c	Dio_config.c
port.c	Can_config.c
	Buzzer_config.c
	Light_config.c

Commen folder (all the header (name.h))
Mainapp.h / os.h / servies.h
Comm_manager.h/Device_manager.h
Light.h /Buzzer.h / Door_sensor.h
Dio.h / port.h / timer.h /can.h/adc.h
dio_config.h/port_config.h / timer_config.h /can_config.h /adc_config.h
Stdtypes.h /common_macro.h /Hw.h