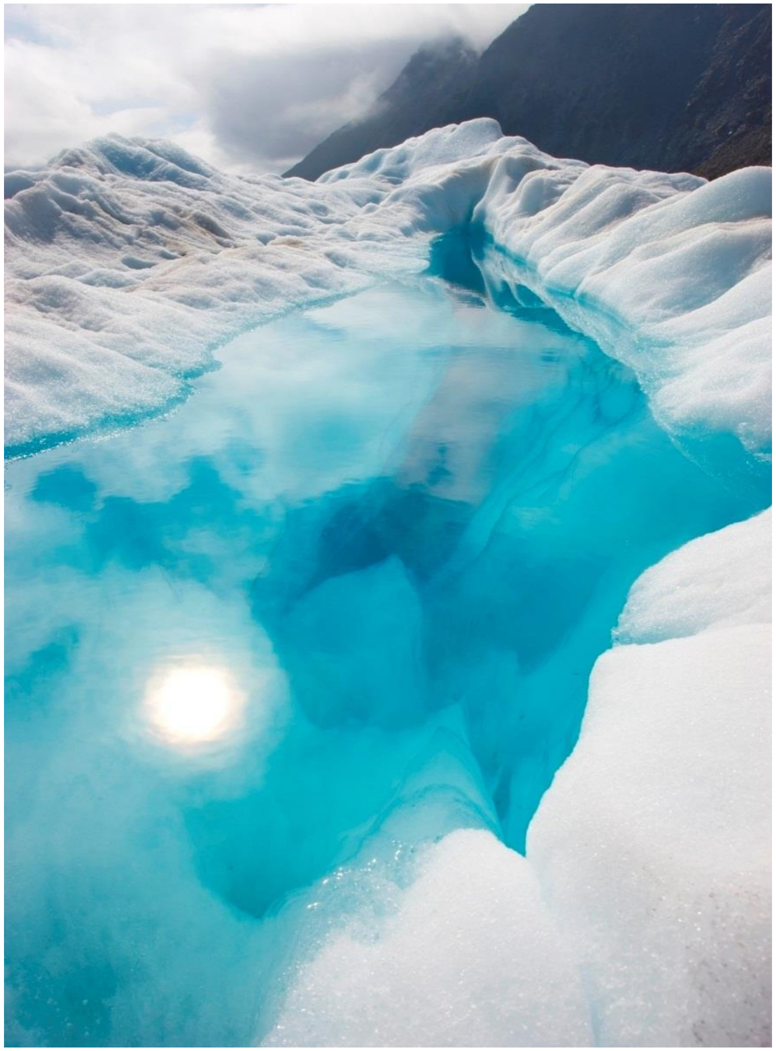
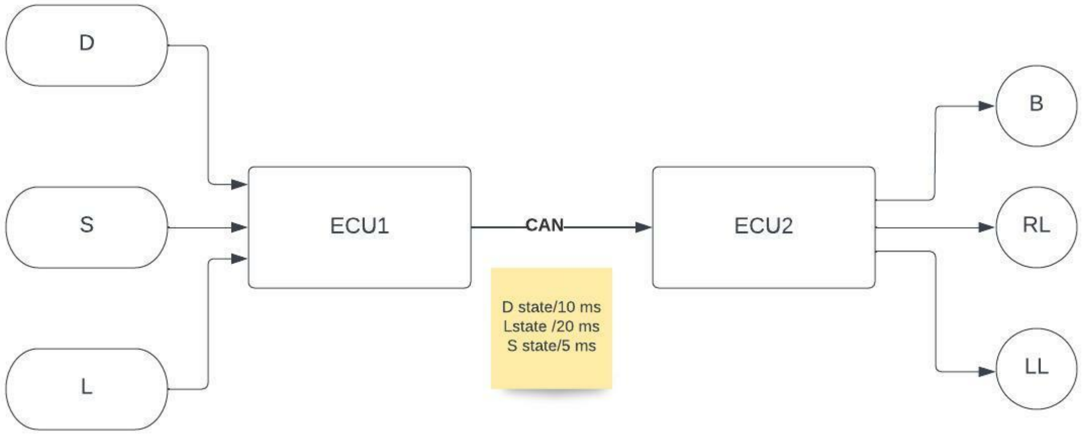


Automotive Door Control System Design

STATIC DESIGN

Nora El-Hennawy | EGYFWD | 4-10-22

System Block Diagram



D state/10 ms
Lstate /20 ms
S state/5 ms

D :Door Sensor

S: Speed Sensor

L: Light Switch

B: Buzzer

RL: Right Light

LL: Left Light

ASSUMPTIONS:

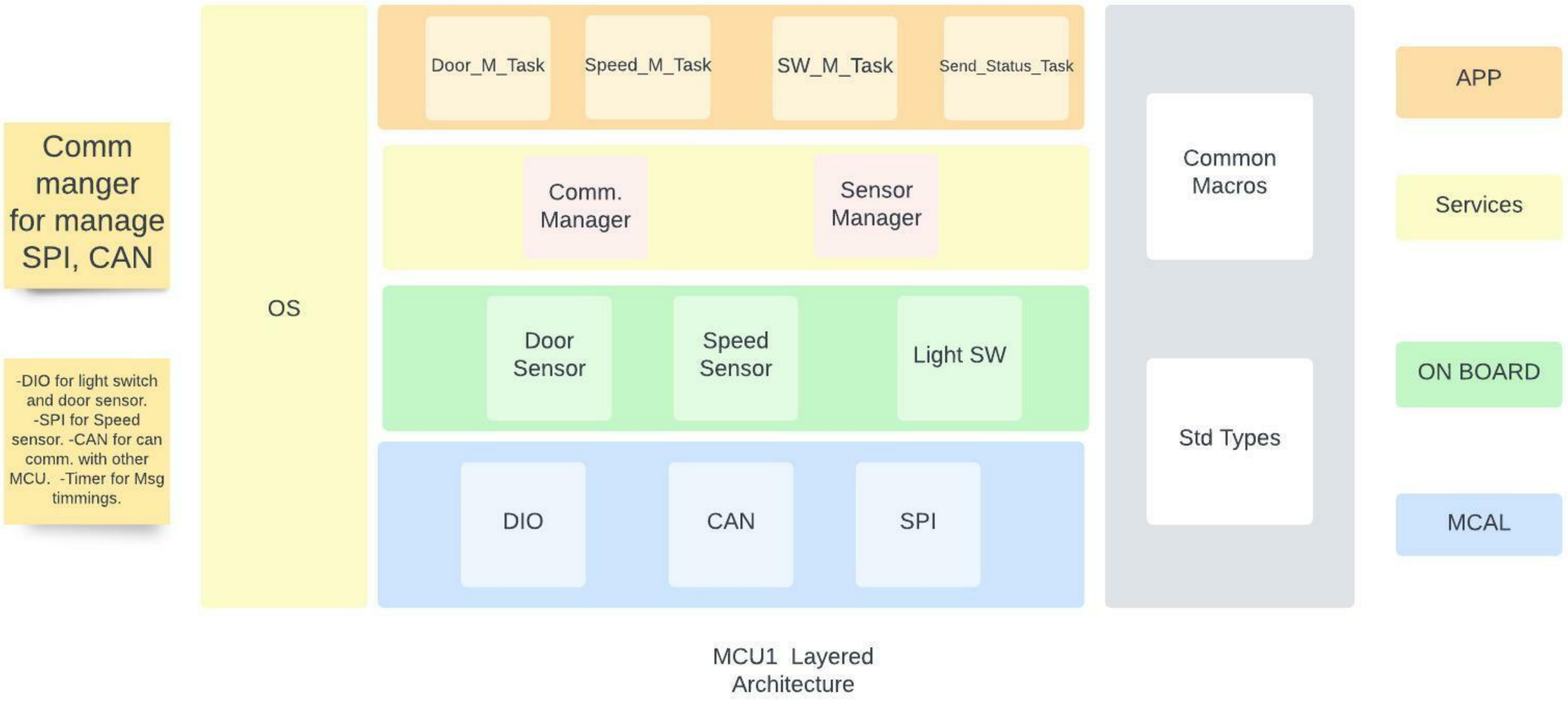
ECU1 has an input capture unit that will be used to take readings from speed sensors and send it on SPI??

DIO Ports on ECU are 16 Pin/Port.

NOTE: If OS used provide Delay function no need for Timer driver.

ECU1

LAYERED ARCHITECTURE DIAGRAM


Comm manger for manage SPI, CAN
-DIO for light switch and door sensor. -SPI for Speed sensor. -CAN for can comm. with other MCU. -Timer for Msg timmings.
OS
Door_M_Task
Speed_M_Task
SW_M_Task
Send_Status_Task
Comm. Manager
Sensor Manager
Door Sensor
Speed Sensor
Light SW
DIO
CAN
SPI
Common Macros
Std Types
APP
Services
ON BOARD
MCAL
MCU1 Layered Architecture

ECU1 COMPPONENTS AND MODULES

MCAL LAYER:

DIO

Driver interface with ECU DIO offering read, write and configure DIO APIs to upper layer.

CAN

Driver interface with ECU CAN Controller to enable read, write on CAN BUS also allow configuration of CAN communication scheme through CAN_INIT() API .

SPI

Driver interface for SPI communication on ECU offering read, write API's and also SPI configuration through SPI_Init().

Timer

Driver for ECU timers offering Delay function and timer configuration function, Since System is assumed to work using operating system thus delay function are provided through OS so this driver is not needed in design.

ON BOARD LAYER:

SPEED_Sensor

Module responsible for interface with speed sensor providing API to set sensor configuration(To determine sensor actually connected to which DIO) and API to get sensor status.

Light_SW

Module responsible for interface with Light Switch providing API to set Light switch configuration(To determine switch actually connected to which DIO) and API to get switch status.

Door_Sensor

Module responsible for interface with speed sensor providing API to set sensor configuration (Set SPI configuration used to communicate with sensor) and API to get sensor status.

SERVICES LAYER:

Communication Manager

Module responsible for managing communication on ECU through different communication types(i.e. CAN ,SPI,..)

The role of this module is to abstract "Application layer" form communication details underlying in" hardware layer" allowing ease to add and/or remove different communication types to the system.

Sensor Manager

Module responsible for managing all system sensors with various types(i.e. DOOR Sensor, speed sensor,light switch..)

The role of this module is to abstract "Application layer" form sensor interface details underlying in" hardware layer" allowing ease to add and/or remove different sensor types to the system.

APPLICATION LAYER:

Tasks:

The system has 4 tasks and 1 queue:

Door Monitor Task, Speed Sensor Monitor Task, And Light Switch Monitor Task:

Each task is responsible for monitoring sensor status for sensor assigned to it and then adding sensor status to the "Status Queue" at preset intervals as stated in system requirement (SRS).

Status Send Task:

Responsible for extracting sending system status from Status Queue and then send it to ECU 2 through CAN.

Status Queue: Intercommunication to sync status messages form the monitor tasks to Send task.

ECU1 API FUNCTION DESCRIPTION

MCAL LAYER

DIO:

Function name: void API_DIO_Init(void)			
Arguments	Inputs: None	A1:	Type:
		Description:	
		A2:	Type
		Description:	
Return	R:	Type:	
	Description:		
Description	Initialize DIO driver with user configuration through calling DIO_Cnfg.c API		

Function name: DIOPinVal API_DIO_ReadPin(DIOPinID PinNum, DIOPortID PortID)			
Arguments	Inputs: PinNum ,PortID	A1: PinNum	Type: DIOPinID
		Description: Pin number to read.	
		A2:PortId	Type: DIOPortID
		Description: ID for Port that contains PinNum	
Return	R: PinVal	Type: DIOPinVal	
	Description: Pin value for PinNum on PortID		
Description	Read Value fro PinNum on PortID		

Function name: DIOPortVal API_DIO_ReadPort(DIOPortID PortID)			
Arguments	Inputs: PortID	A1: PortId	Type: DIOPortID
		Description: ID for Port to read from.	
		A2:	Type:
		Description:	
Return	R: PortVal	Type: DIOPortVal	
	Description: current reading of Port value for PortID		
Description	Read Value for PortId		

Function name: void API_DIO_WritePort(DIOPortID PortID, DIOPortVal PortVal)			
Arguments	Inputs: PortID , PortVal	A1: PortId	Type: DIOPortID
		Description: ID for Port to write on.	
		A2:PortVal	Type: DIOPortVal
		Description: Value to write on Port PortID.	
Return	R: None	Type:	
	Description:		
Description	Writes Value :PortVal on Port: PortId		

Function name:	**void API_DIO_WritePin(DIOPortID PortID, DIOPinID PinNum, DIOPinVal PinVal)**		
Arguments	Inputs: PortID ,PinNum, PinVal	A1: PortId	Type: DIOPortID
		Description: ID for Port to write on.	
		A2:PinNum	Type: DIOPinID
		Description: PinNum to write on Port: PortID.	
		A3:PinVal	Type: DIOPinVal
		Description: Value to write on Pin :PinNum on Port: PortID.	
Return	R: None	Type:	
	Description:		
Description	Writes Value :PinVal on Pin:PinNum on Port: PortId		

Function name:	**void API_DIO_TogglePin(DIOPinID PinNum, DIOPortID PortID)**		
Arguments	Inputs: PinNum ,PortID	A1: PinNum	Type: DIOPinID
		Description: Pin number to toggle.	
		A2:PortId	Type: DIOPortID
		Description: ID for Port that contains PinNum	
Return	R: None	Type:	
	Description:		

Description	Toggle PinNum on PortID

TYPE DEFINES:

- DIOPinVal :

  DataType: u8 **Description**: DIO Pin value (0/1)

- DIOPortVal :

  DataType: u8 **Description**: DIO Port value (0x00→0xff)

- DIOPinID :

  DataType: u8 **Description**: DIO Pin number (0 → 8)

- DIOPortID :

  DataType: u16 **Description**: DIO Port Id (PortA,PortB....)

CAN

Function name: void API_CAN_Init(void)			
Arguments	Inputs: None	A1:	Type:
		Description:	
		A2:	Type
		Description:	
Return	R:	Type:	
	Description:		
Description	Initialize CAN driver with user configuration through calling CAN_Cnfg.c API		

Function name: void API_CAN_Write(CAN_Msg Msg)			
Arguments	Inputs: Msg	A1:Msg	Type: CANMsg
		Description: Message to send on Can Bus	
		A2:	Type
		Description:	
Return	R:	Type:	
	Description:		
Description	Send message “Msg” on Can Bus		

Function name: u8 API_CAN_Read(CAN_Msg * Msg)			
Arguments	Inputs: Msg	A1:Msg	Type:
		Description: pointer to can message structure to return	

		message read from can bus in it.	
		A2:	Type
		Description:	
Return	R: NewMsg	Type:Bool	
	Description: return True if new message was received from can bus else return False.		
Description	Fill” Msg” with new message received from can bus else return False.		

TYPE DEFINES:

CAN_Msg

Type: Structure   Description: Structure containing CAN message (vary according to used Can protocol)

SPI

Function name:	**void API_SPI_Init(void)**		
Arguments	Inputs: None	A1:	Type:
		Description:	
		A2:	Type
		Description:	
Return	R:	Type:	
	Description:		
Description	Initialize SPI driver with user configuration through calling SPI_Cnfg.c API to load SPI configuration entered by user		

Function name:	**void API_SPI_Write(SPI_Msg Msg)**		
Arguments	Inputs: Msg	A1:Msg	Type: SPI_Msg

		Description: Message to send on SPI	
		A2:	Type
		Description:	
Return	R:	Type:	
	Description:		
Description	Send message “Msg” on SPI		

Function name: u8 API_SPI_Read(SPI_Msg * Msg)			
Arguments	Inputs: Msg	A1:Msg	Type:
		Description: pointer to SPI message structure to return message read from SPI in it.	
		A2:	Type
		Description:	
Return	R: NewMsg	Type:Bool	
	Description: return True if new message was received from SPI bus else return False.		
Description	Check if new message was received from SPI Fill” Msg” and return True else return False.		

TYPE DEFINES:

SPI_Msg

Type: Structure Description: Structure containing SPI message.

TIMER

Function name: void API_Timer_Init(void)			
Arguments	Inputs: None	A1:	Type:
		Description:	
		A2:	Type
		Description:	
Return	R:	Type:	
	Description:		
Description	Initialize Timer driver with user configuration		

Function name: void API_SetTimerCallBack(FncPtr* CallBackFnc)			
Arguments	Inputs: CallBackFnc	A1:CallBackFnc	Type: pointer to function
		Description: pointer to call back function to call in Timer ISR	
		A2:	Type
		Description:	
Return	R:	Type:	
	Description:		
Description	Assign call back function to be called in Timer ISR		

Function name: void API_DelayMs(u16 Delay)			
Arguments	Inputs: Delay	A1: Delay	Type: u16
		Description: Value of delay to wait in milliseconds	

		A2:	Type
		Description:	
Return	R:	Type:	
	Description:		
Description	Wait for (Delay) ms		

NOTE: If OS used provide Delay function no need for Timer driver.

ON BOARD LAYER:

SPEED_SENSOR

Function name: void SSensor_Init(void)			
Arguments	Inputs: None	A1:	Type:
		Description:	
		A2:	Type
		Description:	
Return	R: None	Type:	
	Description:		
Description	This function is called at system startup to Initialize Speed Sensor driver with user configuration through calling SpeedSensor_Cnfg.c API to load Speed sensor configuration entered by user		

Function name: S_SensorState API_GetSpeedSensorState(SensorID SensorNum)			
Arguments	Inputs: SensorNum	A1: SensorNum	Type: SensorID

		Description: speed sensor number.	
		A2:	Type
		Description:	
Return	R: SensorState	Type:S_SensorState	
	Description: Speed Sensor State		
Description	Return Speed Sensor State for given sensor number		

TYPE DEFINES:

- SensorID

  Type: enum   Description: unique identifier for each sensor

- S_SensorState

  Type: enum   Description: unique identifier for speed sensor states(CAR_IS_MOV,CAR_STOPPED)

LIGHT_SW

Function name: void LightSW_Init(void)			
Arguments	Inputs: None	A1:	Type:
		Description:	
		A2:	Type
		Description:	
Return	R: None	Type:	
	Description:		

Description	This function is called at system startup to Initialize LightSw module with user configuration through calling LightSW_Cnfg.c API to load LightSW configuration entered by user

Function name: L_SW_State API_Get_L_SWState(L_SW_ID L_SW_Num)			
Arguments	Inputs: L_SW_Num	A1: L_SW_Num	Type: L_SW_ID
		Description: Light SW number.	
		A2:	Type
		Description:	
Return	R: SwState	Type: L_SW_State	
	Description: Light switch State		
Description	Return Light switch State for given light switch number		

TYPE DEFINES:

- L_SW_Num

  Type: enum Description: unique identifier for each light switch

- L_SW_State

  Type: enum Description: unique identifier for light switch states(SW_PRESSED,SW_RELEASED)

DOOR_SENSOR

Function name:	**D_SensorState API_GetDoorSensorState(SensorID SensorNum)**		
Arguments	Inputs: SensorNum	A1: SensorNum	Type: SensorID
		Description: door sensor number.	
		A2:	Type
		Description:	
Return	R: SensorState	Type:S_SensorState	
	Description: Door Sensor State		
Description	Return Door Sensor State for given sensor number		

TYPE DEFINES:

- SensorID

  Type: enum Description: unique identifier for each door sensor

- D_SensorState

  Type: enum Description: unique identifier for door sensor states(DOOR_OPEN,DOOR_CLOSED)

SERVICE LAYER

SENSOR MANAGER

Function name:	**SensorState API_ Get_Sensor_Status (SensType SensorType)**		
Arguments	Inputs: SensorType	A1: SensorNum	Type: SensorID

		Description: speed sensor number.	
		A2: SensorType	Type: SensType
		Description: Sensor Type to get status for	
Return	R: SensorState	Type: void*	
	Description: Sensor State		
Description	Return Sensor State for given sensor type and number This is supervisor(generic)function to manage all system sensor with different types		

TYPE DEFINES:

SensType

Type: enum Description: unique identifier for each sensor type in system(SPEEDSENSOR=0,DOORSENSOR=1,…)

- SensorState

Type: enum Description: unique identifier for sensor states(along with sensor type determine given sensor state)

COMMUNICATION MANAGER

Function name:	**API_Send_Msg (COMM_ID ComType , Msg* Message)**		
Arguments	Inputs: ComType	A1: ComType	Type: COMM_ID
		Description: type of communication used to send Message	
		A2:Message	Type void*

		Description: Message to be sent.
Return	R: SendState	Type: SendState
	Description: Send Status (success/fail)	
Description	Send message on given communication type and return sending status.	

Function name:	**API_Rcev_Msg (COMM_ID ComType , Msg* Message)**		
Arguments	Inputs: ComType	A1: ComType	Type: COMM_ID
		Description: type of communication used to receive Message	
		A2:Message	Type void*
		Description: Message received.	
Return	R: R_Status	Type: RcevState	
	Description: Receive Status (new message/no message)		
Description	Receive message on given communication type and return receive status.		

TYPE DEFINES:

- COMM_ID

  Type: enum Description: unique identifier for each communication type(CAN=0,SPI=1,….)

- SendState

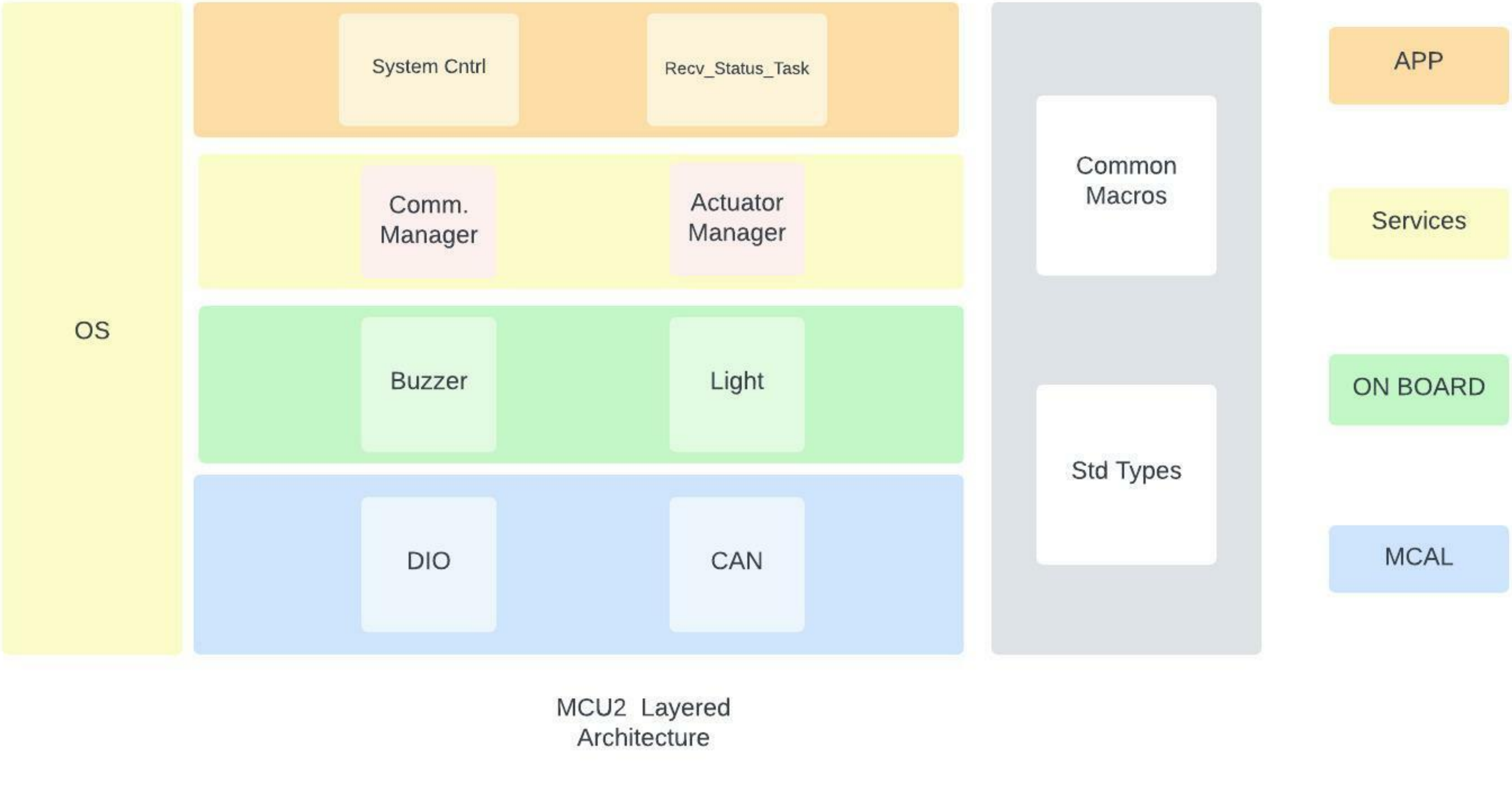
  Type: enum Description: unique identifier for sending status(FAIL=0,SUCCESS=1)

ECU1 FOLDER STRUCTURE

Attached with project.

ECU2

LAYERED ARCHITECTURE DIAGRAM



ECU2 COMPPONENTS AND MODULES

MCAL LAYER:

DIO

Driver interface with ECU DIO offering read, write and configure DIO APIs to upper layer.

CAN

Driver interface with ECU CAN Controller to enable read, write on CAN BUS also allow configuration of CAN communication scheme through CAN_INIT() API .

Timer

Driver for ECU timers offering Delay function and timer configuration function, Since System is assumed to work using operating system thus delay function are provided through OS so this driver is not needed in design.

ON BOARD LAYER:

BUZZER

Module responsible for interface with BUZZER providing API to set BUZZER configuration (To determine BUZZER actually connected to which DIO) and API to turn buzzer on/off. The module allows addition/removal of extra buzzers to system.

LIGHT

Module responsible for interface with system Light Switch providing API to set Light configuration (To determine light actually connected to which DIO) and API to turn light on/off.

The module allows control of different lights on system (front left, front right, side left, side right, back left…etc).

SERVICES LAYER:

Communication Manager

Module responsible for managing communication on ECU through different communication types (i.e. CAN ,SPI,..)

The role of this module is to abstract “Application layer” form communication details underlying in” hardware layer” allowing ease to add and/or remove different communication types to the system.

Actuator Manager

Module responsible for managing all actuators available in the system (i.e. Buzzer, Light..)

The role of this module is to abstract “Application layer” form sensor interface details underlying in” hardware layer” allowing ease to add and/or remove different actuator types to the system.

APPLICATION LAYER:

Tasks:

The system has 2 tasks and 1 queue:

System control Task:

Each task is responsible for updating system status from status queue and then taking action on Buzzer and lights sensor accordingly (as stated in system requirement (SRS)).

Status Receive Task:

Responsible for receiving system status from ECU1 on CAN and adding it to Status Queue to be further processed through System control task.

Status Queue: Intercommunication to sync status messages form the receive task to System control task.

ECU2 API FUNCTION DESCRIPTION
DIO ,CAN ,COMM_MANAGER

Refer to same modules in ECU1 API Function description

BUZZER

Function name: void BUZZER_Init(void)			
Arguments	Inputs: None	A1:	Type:
		Description:	
		A2:	Type
		Description:	
Return	R: None	Type:	
	Description:		
Description	This function is called at system startup to set configuration of all system buzzers and set initial state to all buzzers at system startup.(i.e. initial state for all system buzzer to be off at start up for example). Note:Buzzer configuration is done by user in buzzer_cnfg.h and Buzzer_CNFG.c ,Buzzer_init function only load this configuration through buzzer_cnfg.c API.		

Function name: BUZZER_ON(BUZZER_ID B_Num)			
Arguments	Inputs: B_Num	A1: B_Num	Type: BUZZER_ID
		Description: Buzzer number to turn ON.	
		A2:	Type
		Description:	

Return	R:	Type:
	Description:	
Description	Turn Buzzer number :B_Num ON.	

Function name: BUZZER_OFF(BUZZER_ID B_Num)			
Arguments	Inputs: B_Num	A1: B_Num	Type: BUZZER_ID
		Description: Buzzer number to turn OFF.	
		A2:	Type
		Description:	
Return	R:	Type:	
	Description:		
Description	Turn Buzzer number :B_Num OFF.		

TYPE DEFINES:

- BUZZER_ID

  Type: enum Description: unique identifier for each buzzer

LIGHT

Function name: void Light_Init(void)			
Arguments	Inputs: None	A1:	Type:
		Description:	

		A2:	Type
		Description:	
Return	R: None	Type:	
	Description:		
Description	This function is called at system startup to set configuration of all system light and set initial state to all lights at system startup according to SRS (i.e. initial state for all system light maybe off at start up for example). Note :Light configuration is done by user in Light_cnfg.h and Light_CNFG.c ,Light_init function only load this configuration through Light_cnfg.c API.		

Function name:	**Light_ON(LIGHT_ID L_ID)**		
Arguments	Inputs: L_Num	A1: L_Num	Type: LIGHT _ID
		Description: Light ID to turn ON.	
		A2:	Type
		Description:	
Return	R:	Type:	
	Description:		
Description	Turn Light with ID= L_ID ON.		

Function name:	**Light_OFF(LIGHT_ID L_ID)**		
Arguments	Inputs: L_Num	A1: L_Num	Type: LIGHT _ID

		Description: Light ID to turn OFF.	
		A2:	Type
		Description:	
Return	R:	Type:	
	Description:		
Description	Turn Light with ID= L_ID OFF.		

TYPE DEFINES:

- LIGHT_ID

  Type: enum   Description: unique identifier for each buzzer(LEFT_FRONT_LIGHT=0,RIGHT_FRONT_LIGHT=1,.....)

ACTUATOR MANAGER

Function name:	**API_ Set_Actuator (ActType Actuator, Act_ID Act_num , ACTION_ID Action)**		
Arguments	Inputs: Actuator , Act_num, Action	A1: Actuator	Type: ActType
		Description: Type of Actuator to set.	
		A2: Act_num	Type: Act_ID
		Description: Actuator number to SET	
		A3:Action	Type:Action_ID
		Description:Action done on actuator(ON/OFF)	
Return	R: SensorState	Type:	
	Description: Sensor State		
Description	This is supervisor(generic)function to manage all system actuators with different types		

TYPE DEFINES:

ActType

Type: enum   Description: unique identifier for each actautor type in system(BUZZER=0,LIGHT=1,…)

Action_ID

Type: enum   Description: unique identifier for action to take place (OFF=0,ON=1)

SYSTEM CNTRL

Function name: UpdateSystemStatus(StatusMsg)			
Arguments	Inputs: StatusMsg	A1: StatusMsg	Type: STAT_MSG
		Description: Light ID to turn ON.	
		A2:	Type
		Description:	
Return	R:	Type:	
	Description:		
Description	Update system status(Door,LightSW,Car_State) according to status message (extracted form status queue)		

Function name: SystemCtrl()			
Arguments	Inputs:	A1:	Type:
		Description: Light ID to turn ON.	
		A2:	Type
		Description:	
Return	R:	Type:	
	Description:		
Description	State machine that takes action according to current system status. (Door,LightSW,Car_State)		

TYPE DEFINES:

STAT_MSG

Type: structure Description: structure for sensor status message received for ECU1.

ECU2 FOLDER STRUCTURE

Attached with project CTOS ECU2 folder