Automotive Door Control System Design

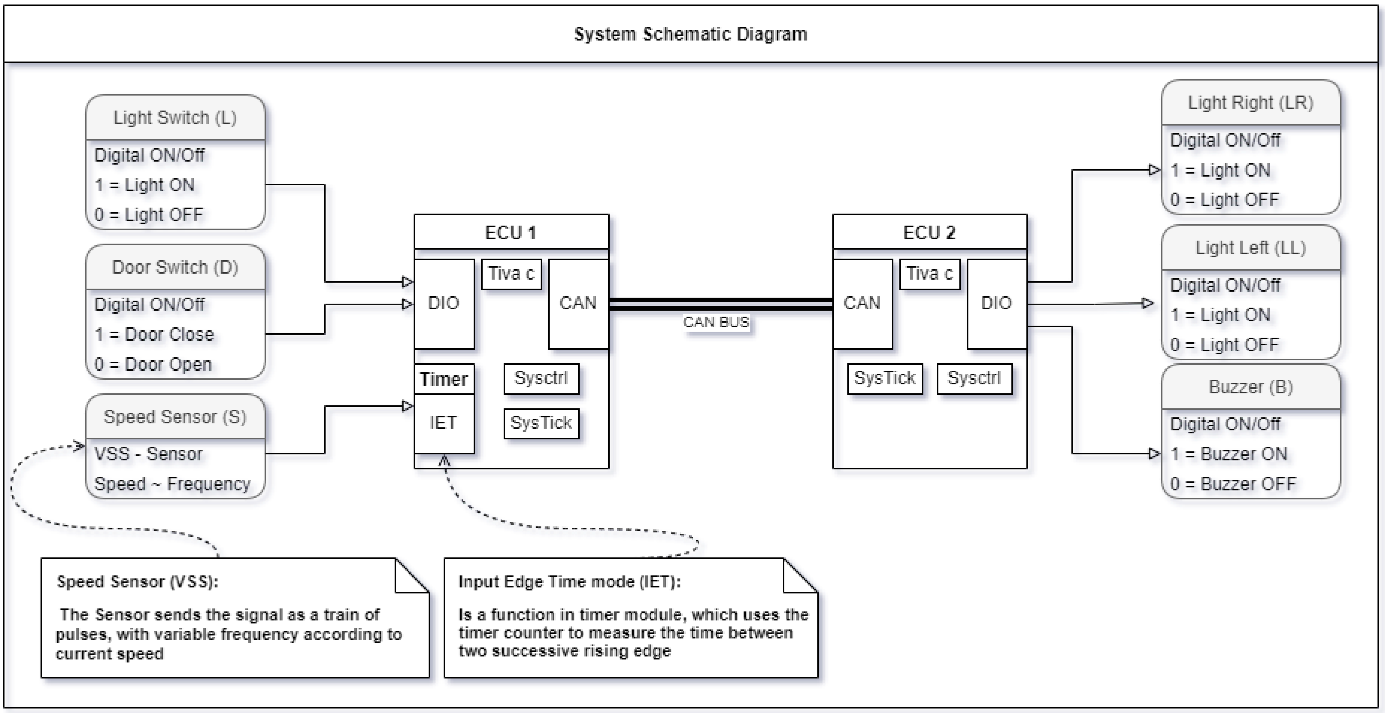
Static Design

Mohamed Abdullah Mohamed Hassan

EG-FWD Embedded System Design
September cohort

Table of Contents

I. System Schematic:



Speed Sensor (VSS):

The Sensor sends the signal as a train of pulses, with variable frequency according to current speed

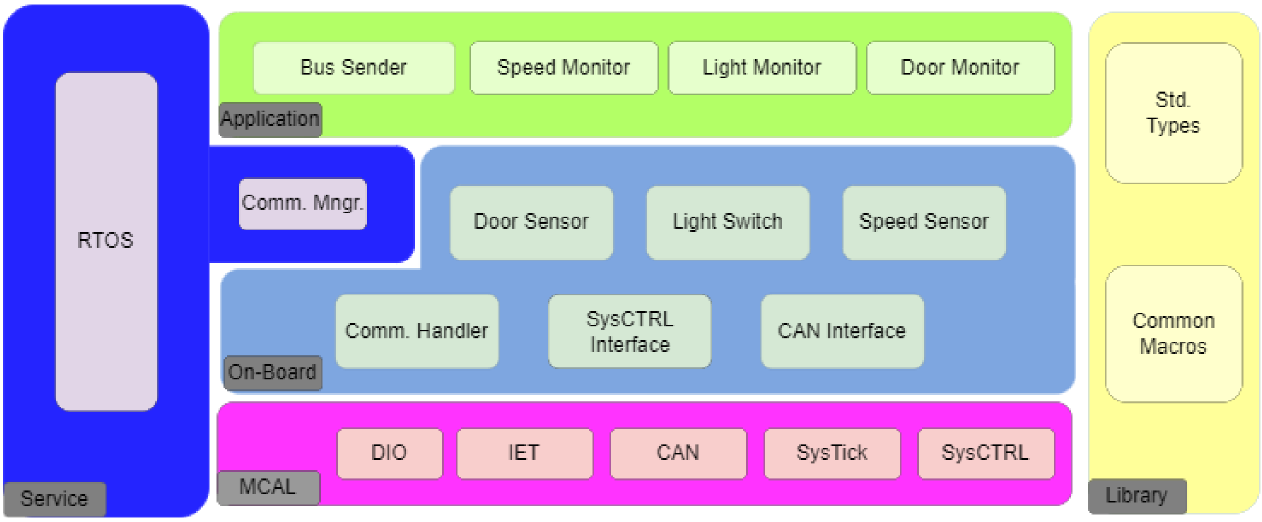
Input Edge Time mode (IET):

Is a function in timer module, which uses the timer counter to measure the time between two successive rising edge

II. Project Static Design

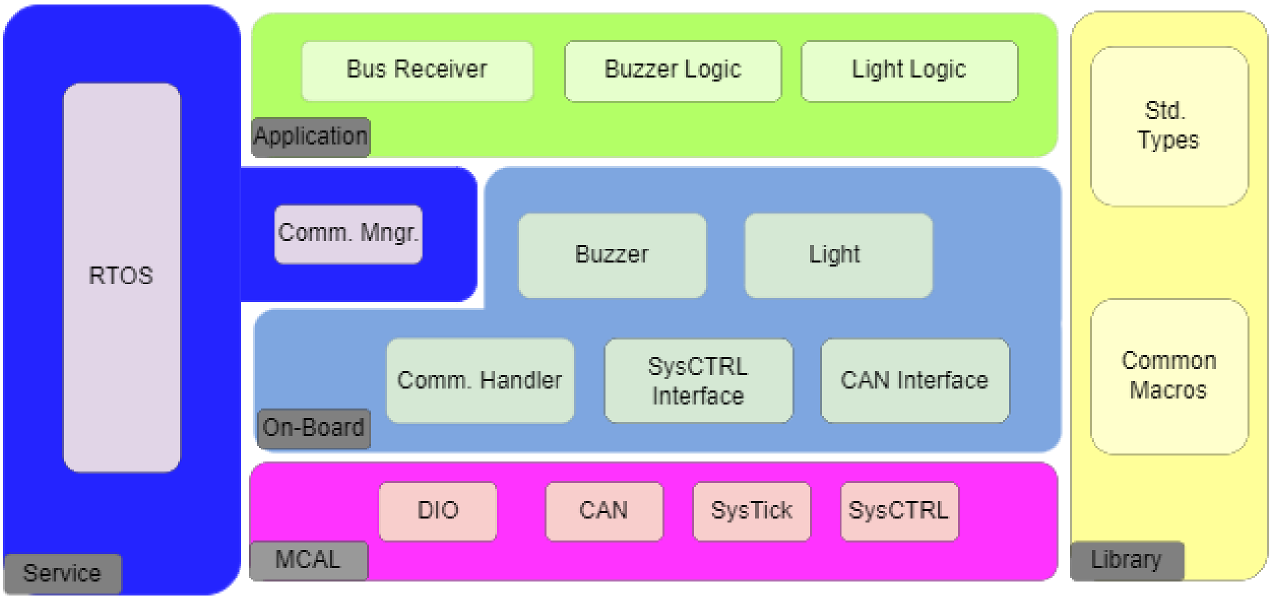
A. Layered architecture:

1. ECU 1



ECU 1 Layered arch.

2. ECU 2



ECU 2 Layered arch.

B. MCAL APIs

1. Digital Input Output Module “DIO”

void DIO_Setup (Digital_Pin_t * pin)	**Module: DIO**
Description: - Enable Clock for the Port - Configure the Pin for Digital Input, and Output	
Pointer to Digital_Pin_t struct	
Return: No return	

xbool_t DIO_Read (Digital_Pin_t * pin)	**Module: DIO**
Description: - Returns the current state of the pin	
Input parameters: Pointer to Digital_Pin_t struct	
Return: xFlase = Pin is low xTrue = Pin is High	

void DIO_Write (Digital_Pin_t *pin, xbool_t state)	**Module: DIO**
Description: Set the PIN satatus according to state	
Input parameters: - Pointer to Digital_Pin_t struct - State of pin High, Low	
Return: No return	

2. System Timer Module “SYS Tick”

void SysTick_Setup (uint32_t reloadValue, void * callBackFunction)	**Module: SYS Tick**
Description: Selects the clock source Setup the timer start value	
Input parameters: reloadValue the start value of the timer pointer for callback function which will be called on ISR	
Return: No return	
Notes: This Method Doesn’t Start the SYS Tick timer module. If callBackFunction is NULL the Interrupt enable bit will be disabled.	

Void SysTick_Sart (void)	**Module: SYS Tick**
Description: Starts the System Tick module after setup.	
Input parameters: No Input	
Return: No Return	
Notes: Calling SysTick_Setup() is mandatory before calling SysTick_Sart()	

void SysTick_Stop (void)	**Module: SYS Tick**
Description: Hold the System Tick module from counting.	
Input parameters: No Input	
Return: No return	

uint32_t SysTick_Read (void)	**Module: SYS Tick**
Description: Returns the current value of the countdown register.	
Input parameters: No Input	
Return: uint32_t which represents the current value of countdown register.	

3. System Control Module “SysCTRL”

uint32_t SysCTRL_Init (void)	**Module: Sys CTRL**
Description: Configure the PLL with required clock frequency. Configure the MCU to Clock source.	
Input parameters: No Input	
Return: No Return	

4. Input Edge Time Capture Module “IET”

uint32_t IET_Init (void * callBackFunction)	**Module: IET**
Description: Enable & configure the module for Operation Call Back the assigned function upon ISR.	
Input parameters: pointer for callback function which will be called on ISR	
Return: No Return	
Notes: This Method doesn’t Start IET Module	

uint32_t IET_Start (void)	**Module: IET**
Description: Start IET Operation	
Input parameters: No Input	
Return: No Return	
Notes: Calling IET_Ini() is mandatory prior calling this method.	

Uint32_t IET_Stop (void * callBackFunction)	**Module: IET**
Description: Hold the operation of IET Module.	
Input parameters: No Input.	
Return: No Return	

5. Controller Area Network Module “CAN”

xstate_t CAN_Init (can_config_t *config)	**Module: CAN**
Description: Enable CAN module Setup CAN Bit rate Configure CAN RX messages filter	
Input parameters: Pionter to can_config_t struct	
Return: xstate_t Error code	
Notes: This Method Configures the CAN module, and doesn’t connect it to the network	

xstate_t CAN_Open (void)	**Module: CAN**
Description: Starts the CAN module, and connect to the bus for sending and receiving data	
Input parameters: No Input	
Return: xstate_t Error code	
Notes: Calling CAN_Init() is mandatory prior calling CAN_Open()	

xstate_t CAN_Close (void)	**Module: CAN**
Description: Halts the CAN module, and Disconnect from the bus.	
Input parameters: No Input	
Return: xstate_t Error code	

xcan_state_t CAN_Send (com_msg_t *msg)	**Module: CAN**
Description: Sends a CAN message	
Input parameters: Pointer to the message to be sent	
Return: xcan_state_t Error code	

xcan_state_t CAN_Read (com_msg_t *msg)	**Module: CAN**
Description: Reads the received buffer and place it in ca com_msg_t n_msg_t pionter	
Input parameters: Pionter to com_msg_t struct to place the data in	
Return: xcan_state_t Error code	

xcan_state_t CAN_Get_State (void)	**Module: CAN**
Description: Returns The Current Status of the CAN module and CAN bus	
Input parameters: No Input	
Return: xcan_state_t Error code	

C. On-Board APIs:

1. Door module: “door”

xstate_t Door_Init (dio_pin_t *doorPin)	**Module: door**
Description: Start the initialization of door sensor as an Input pin	
Input parameters: Pointer to dio_pin_t struct.	
Return: xstate_t Error code	

xbool_t Door_Get_State (dio_pin_t *doorPin)	**Module: door**
Description: Start the initialization of door sensor as an Input pin	
Input parameters: Pointer to dio_pin_t struct.	
Return: xFlase = Door is closed xTrue = Door is opened	

2. Light Switch Module “light_sw”

xstate_t Lightsw_Init (dio_pin_t *doorPin)	**Module: light_sw**
Description: Start the initialization of light switch as an Input pin	
Input parameters: Pointer to dio_pin_t struct.	
Return: xstate_t Error code	

xbool_t Lightsw_Get_State (dio_pin_t *doorPin)	**Module: light_sw**
Description: Reads the state of the light switch	
Input parameters: Pointer to dio_pin_t struct.	
Return: xFlase = Switch is released xTrue = Switch is Pressed	

3. Vehicle Speed Sensor Module: “vspeed”

xstate_t VSpeed_Init (void)	**Module: vspeed**
Description: Initialize IET module and place call-back function to measure the speed service	
Input parameters: no-Input	
Return: xstate_t Error code	
Note: Only one speed sensor used.	

uint32_t VSpeed_Read (dio_pin_t *doorPin)	**Module: vspeed**
Description: Request to return the current speed of the vehicle.	
Input parameters: No-Input	
Return: 0 = vehicle is stopped 0 >= vehicle is moving, the current speed is returned.	

4. Can bus interface module: “bus can”

xstate_t Bus_CAN_Init (void)	**Module: bus can**
Description: Initialize CAN module, and start receiving	
Input parameters: No-Input	
Return: xstate_t Error code	

xsate_t Bus_CAN_read (com_msg_t *msg)	Module: **bus can**
Description: Reads message by its ID and store it in msg struct.	
Input parameters: pointer to com_msg_t struct to hold the message	
Return: xstate_t	

xsate_t Bus_CAN_Send (com_msg_t *msg)	Module: **bus can**
Description: Sends a message through a CAN bus	
Input parameters: pointer to com_msg_t struct to send	
Return: xstate_t	

xcan_state_t Bus_CAN_Errors (void)	Module: **bus can**
Description: Return the most recent CAN bus Error	
Input parameters: no-parameters	
Return: xcan_state_t	

5. MCU System control Interface: “MSys”

xstate_t MSys_Init (void *tickCallBack)	Module: **MSys**
Description: Initialize MCU clock sources. Configure SysTick module. Assigns the SysTick ISR to tickCallBack() function “for RTOS operation”	
Input parameters: pointer to tickCallBack()	
Return: xstate_t	

6. Board communication Handler: “HCom”

xstate_t HCom_Init (com_msg_t *msg)	Module: **HCom**
Description: Initialize communication channel specified by com_msg_t.ch	
Input parameters: pointer to com_msg_t structure	
Return: xstate_t	

xstate_t HCom_send (com_msg_t *msg)	Module: **HCom**
Description: Sends the message through comm interface	
Input parameters: pointer to com_msg_t structure	
Return: xstate_t	

xstate_t HCom_Receive (com_msg_t *msg)	Module: **HCom**
Description: Reads the message from comm interface	
Input parameters: pointer to com_msg_t structure	
Return: xstate_t	

void HCom_Error (string * error_no)	Module: **HCom**
Description: Sends the message through comm interface	
Input parameters: pointer to string to store the error message.	
Return: xstate_t	

7. Light output Module “light”

xstate_t Light_Init (dio_pin_t *doorPin)	Module: **light**
Description: Start the initialization of light as an output pin	
Input parameters: Pointer to dio_pin_t struct.	
Return: xstate_t Error code	

xstate_t Light_Set_State (dio_pin_t *doorPin)	Module: **light**
Description: Sets the desired pin state High, or Low	
Input parameters: Pointer to dio_pin_t struct.	
Return: xstate_t	

8. Buzzer Module “Buzz”

xstate_t Buzz_Init (dio_pin_t *doorPin)	Module: **Buzz**
Description: Start the initialization of Buzzer pin as an output	
Input parameters: Pointer to dio_pin_t struct.	
Return: xstate_t Error code	

xstate_t Buzz_Set_State (dio_pin_t *doorPin)	Module: **Buzz**
Description: Sets the desired pin state High, or Low	
Input parameters: Pointer to dio_pin_t struct.	
Return: xstate_t	

D. ECU 1 Application Tasks:

1. Speed Monitor Task

void uSpeed_Monitor (void * pvParameters)	**Layer: Application**
Description: This Task is responsible for reading, & reporting the vehicle speed	
Setup: • Initialize the Speed_Sensor module	
Loop: • Reads the current speed from speed module. • Converts the speed reading to Km/h. • Sends through RTOS queue the speed value along with task tag ‘S’	
Used API’s: VSpeed_Init() For initializing speed module VSpeed_Read() For reading of the current speed value	
Task periodicity: 5ms	

2. Door Monitor Task

void uDoor_Monitor (void * pvParameters)	**Layer: Application**
Description: This Task is responsible for reading, & reporting the Door sensor state.	
Setup: Initialize the Door module, for door pin as DIO input.	
loop: • Reads the door state from door module. • Sends through RTOS queue the door state along with task tag ‘D’	
Used API’s: Door_Init() For initializing Door input pin module Door_Get_State() For reading of the current Door stae.	
Task Periodicity: 10ms	

3. Light Monitor Task

void uLight_Monitor (void * pvParameters)	**Layer: Application**
Description: This Task is responsible for reading, & reporting the Light switch state.	
Setup: Initialize the Lightsw module, for light pin as DIO input	
Loop: • Reads the Light switch state from door module. • Sends through RTOS queue the light state along with task tag ‘L’	
Used API’s: Lightsw_Init() For initializing Door input pin module Lightsw_Get_State() For reading of the current Door stae.	
Task Periodicity: 20ms	

4. Bus Sender Task

void uBus_Sender (void * pvParameters)	**Layer: Application**
Description: This Task is responsible for sending speed value, door state, & light state over client defined communication channel.	
Setup: • Initialize communication channel in RTOS. • setup three types of messages ID “Speed, Door, Light” to be sent.	
Loop: • Listen to queue for new messages. • fitch new messages on the queue. • prepare the message and it ID, using sent “Tag, Value” • Send message over defined communication channel. • loop until no queue available.	

Used API's:

- RTOS_Init_comm_chanel() RTOS method to start initializing defined channel through On-Board handler.
- RTOS_send_comm_chanel() RTOS method to send the message over defined channel through On-Board handler.

Task periodicity: 5ms "as the fastest task periodicity speed monitor is 5ms"

E. ECU 2 Application Task:

1. Bus Receiver Task

void uBus_Receiver (void * pvParameters) **Layer: Application**

Description:

This task is responsible for reading received messages over client defined communication channel, and then storing them in the global variables.

Variables:

- **speed** for storing received speed value to be used by other tasks "protected by semaphore".
- **door** for storing received door state to be used by other tasks "protected by semaphore".
- **light** for storing received light switch state to be used by other tasks "protected by semaphore".

Setup:

- Initialize communication channel in RTOS.
- setup three types of messages ID "Speed, Door, Light" to be filtered by the receiver.

Loop:

- check for new messages received.
- get new message and parse its destination "speed, door, light" using its ID.
- request semaphore key for desired variable to update.
- update the variable value.
- return the semaphore key.
- loop until no new received messages.

Task Periodicity: 5ms

2. Buzzer Logic Task

void uBuzzer (void * pvParameters) **Layer: Application**

Description:

This task is responsible for reading speed, door, light, variables and preform logic operation according to client defined truth table.

Variables:

- **speed** storing speed value. "Protected by semaphore"
- **door** storing door state. "Protected by semaphore"
- **light** storing light switch state. "Protected by semaphore"

First call:

- Initialize Buzzer module, for buzzer pin as DIO output.

Loop:

- request semaphore key for speed value.
- copy speed variable to local variable.
- return semaphore key.
- repeat for door, light variables.
- calculate buzzer state using client truth table.
- update buzzer pin state through buzzer module.

Used API's:

Buzz_Init() For initializing buzzer pin as output.
Buzz_Set_State() for setting new buzzer state to the output pin.

Task Periodicity : 10ms

3. Light Logic Task

void uLight (void * pvParameters)	**Layer: Application**

Description:

This task is responsible for reading speed, door, light, variables and preform logic Light operation according to client defined truth table.

Variables:

- **speed** storing speed value. “Protected by semaphore”
- **door** storing door state. “Protected by semaphore”
- **light** storing light switch state. “Protected by semaphore”
- **light_off** local variable to count time to turn off the light.
- **Light_delayed_off** local Boolean variable to indicate the light is set for delayed turn-off.

Setup:

- setup Light output module for “LL” as output pin.
- setup Light output module for “RL” as output pin.

Loop:

- request semaphore key for speed value.
- copy speed variable to local variable.
- return semaphore key.
- repeat for door, light variables.
- calculate light state using client truth table.
- If the light is set to be delayed turn-off after 3 seconds
  - check both **Light_delayed_off, Light_off** variable for time-expire.
- update light pin state through Light module.

Used API’s:

Light_Init() For initializing buzzer pin as output.
Light_Set_State() for setting new buzzer state to the output pin.

F. Standard Structures & Enumeration

1. Structures

dio_pin_t		
uint8_t	Port	Port number
uint8_t	Pin	Pin Number
uint8_t	Dir	Pin Direction

can_config_t		
uint32_t	Bitrate	CAN Module Bit Rate
uint8_t	MsgCount	the count of message id
uint8_t	MsgId[]	The array for message id

com_msg_t		
uint32_t	Id	message ID used for some comm channels.
uint8_t	ch	Selects the communication channel “UART, SPI, CAN,…”
uint8_t	MsgLength	The Length of the message
uint8_t	Msg[]	The message it self

2. Enumerations: “Type Define”

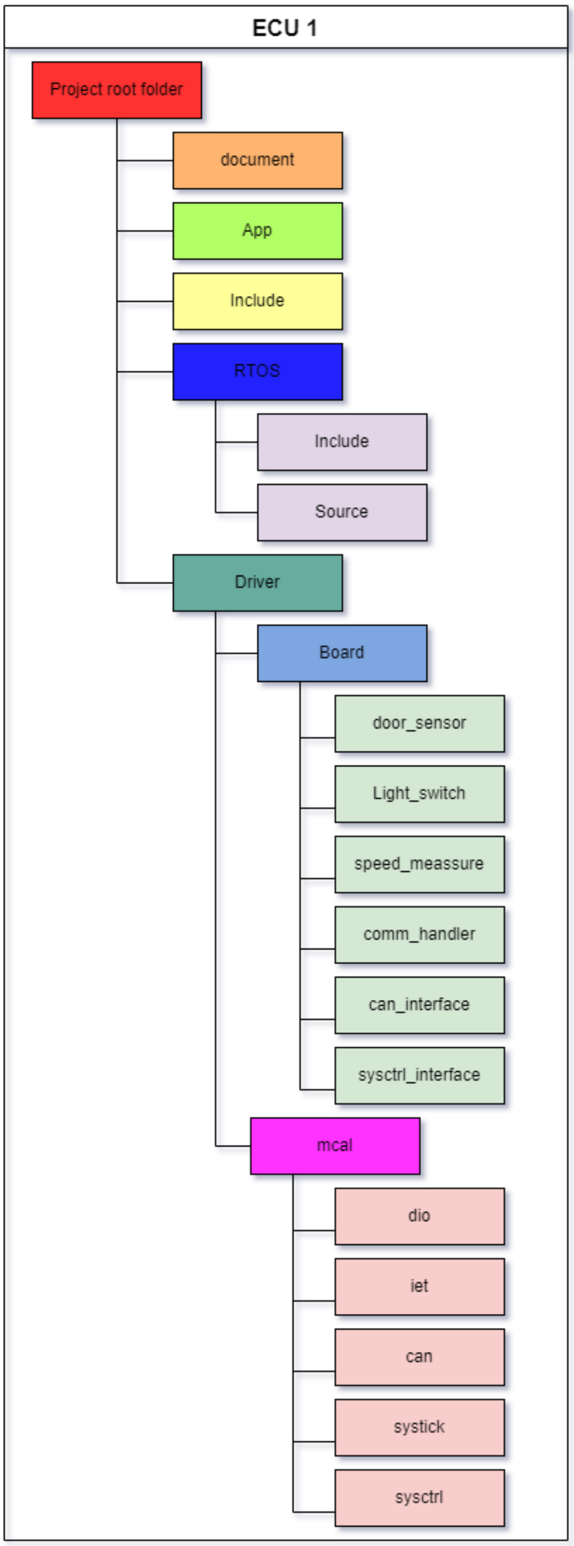
xbool_t		
XFalse	= 0x00	False
XTrue	= !XFalse	True

xcan_bool_t		
xstate_OK	= 0	OK
xstate_Error	= 1	Error

xcan_stsate_t		
xcan_OK	= 0	No Error present
xcan_Error	= 0x01	General Error
xcan_bus_off	= 0x10	CAN module is Disconnected from bus
xcan_tx_overflow	= 0x20	Transmitter Buffer is Full
xcan_buffer_empty	= 0x30	Receiver Buffer is Empty
xcan_msg_empty	= 0x31	No Message by this ID
xcan_rx_overflow	= 0x32	Receiver buffer is full
xcan_bus_warning	= 0x50	Bus error counter is over 96
xcan_bus_error_active	= 0x51	Bus error counter is below 127
xcan_bus_error_pasive	= 0x52	Bus error counter is over 127
xcan_bus_conflict	= 0x53	collusion occurred after arbitration

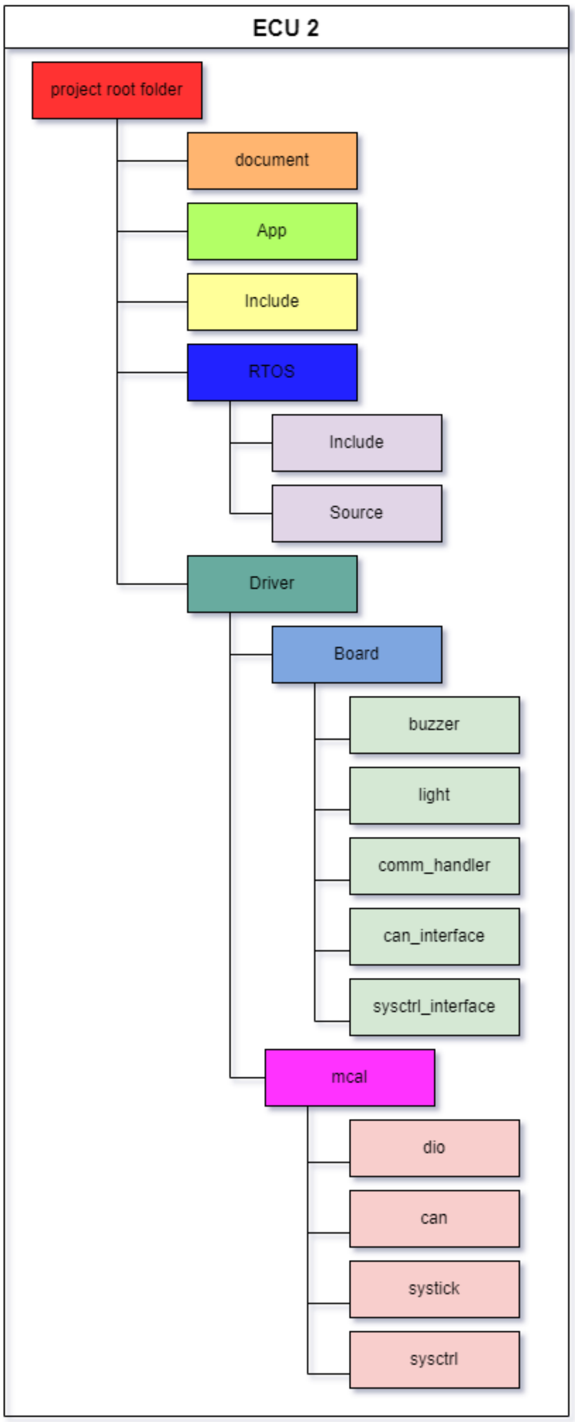
G. Folder Structures:

1. ECU 1 Folder structure:



ECU 1 Project Folder Structure

2. ECU 2 Folder Structure:



ECU 2 Project Folder Structure