Automotive Door Control System Design

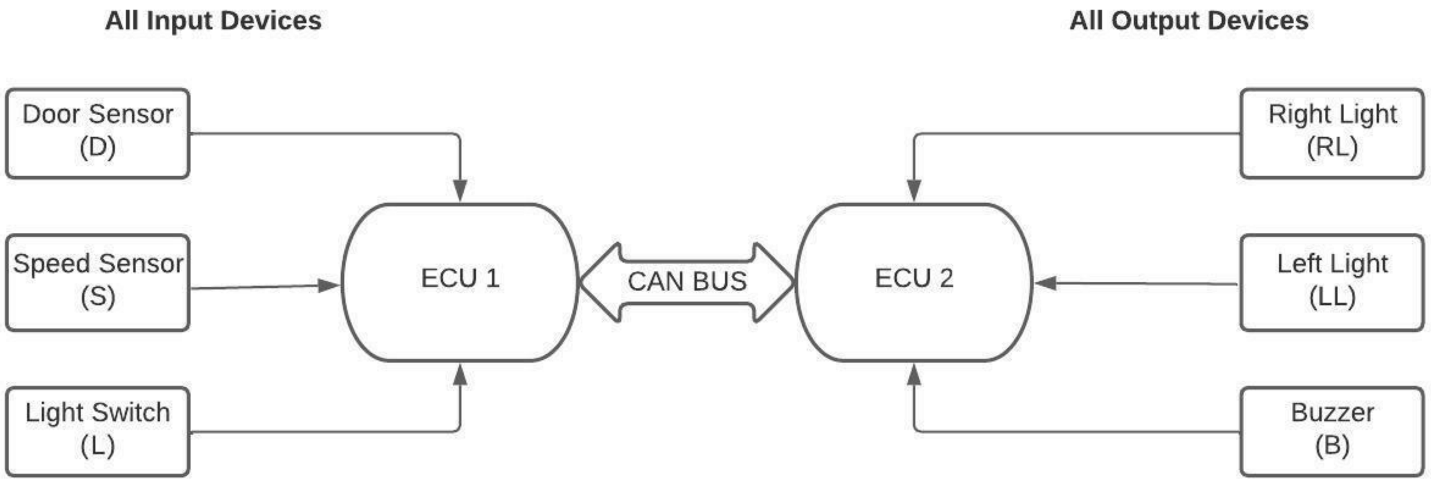
Fully Static Design

Egypt FWD – Advanced Embedded Systems

Hassan Abd El-Kareem Kassem Mostafa

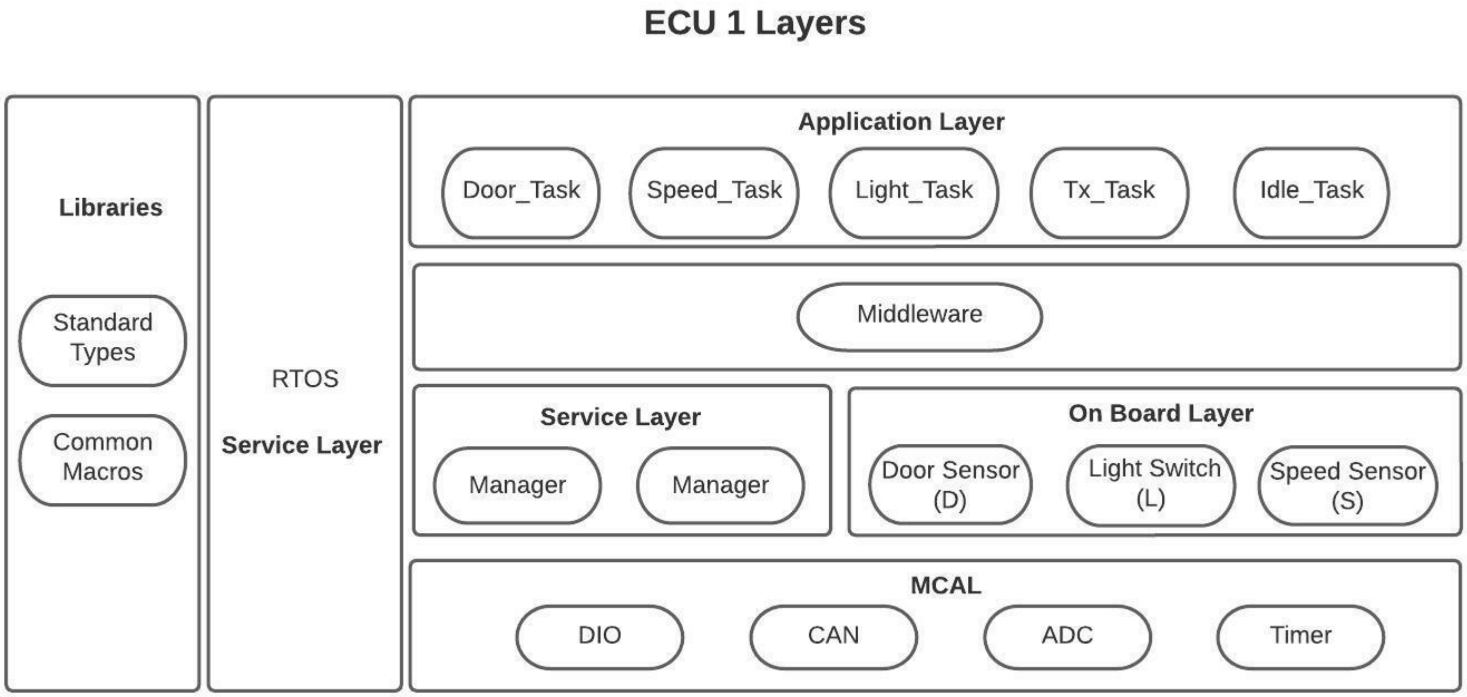
January 1, 2023

I. System Schematic (Block Diagram):



II. ECU 1:

a. The Layered Architecture:



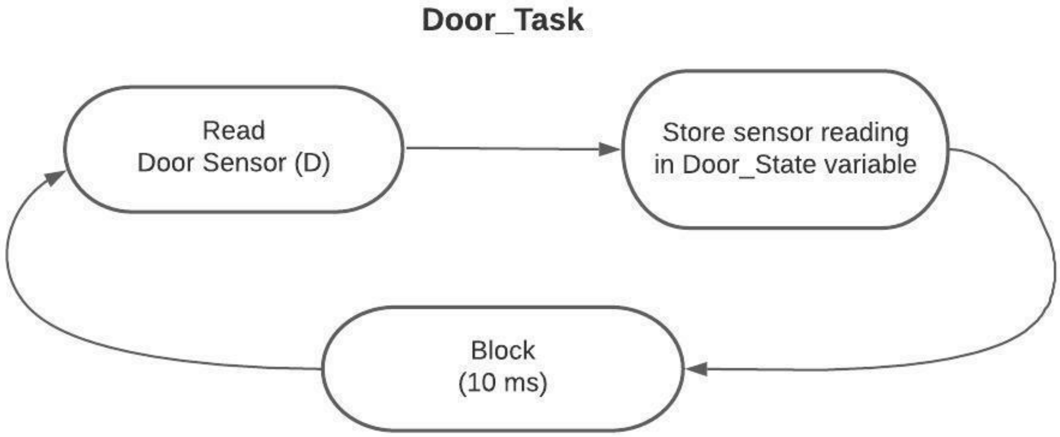
b. Components And Modules:

We'll use Real Time Operating System (RTOS) with ECU 1, as it reads sensors & switch values, calibrates, maps, and prepares data frame, then sends data to ECU 2 through CAN Bus.

i. Door Sensor (D):

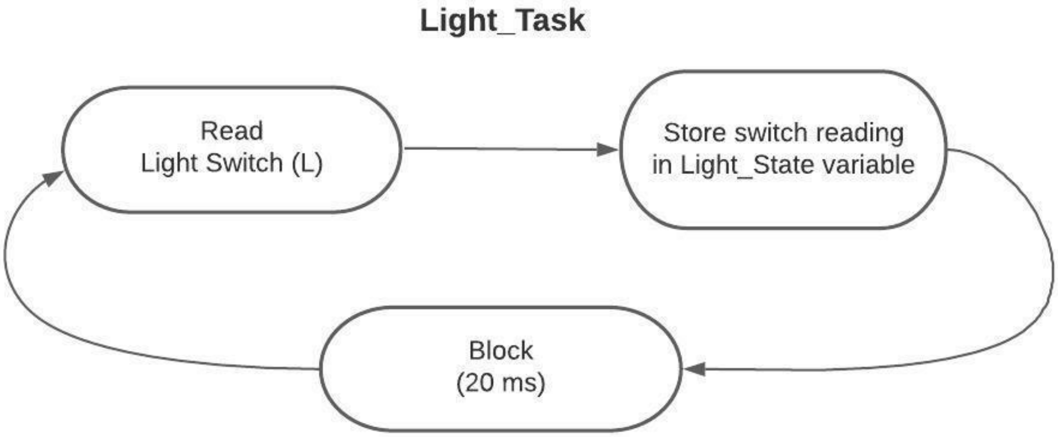
We'll create Door_Task to serve Door Sensor function (as described in flowchart)

as its Periodicity is 10 ms with Priority of 1.



ii. Light Switch (L):

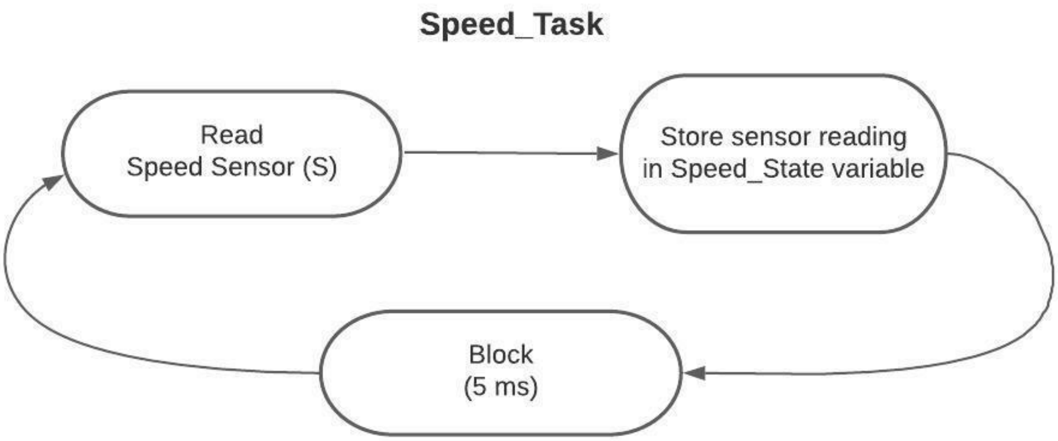
Light_Task will serve light switch function (as follows)



With Periodicity of 20 ms & Priority of 1.

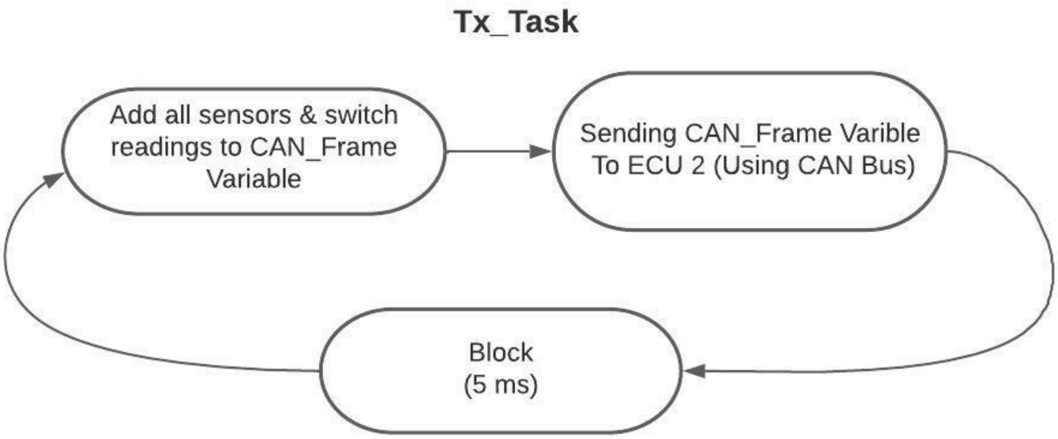
III. Speed Sensor (S):

We’ll use Speed_Task for speed Sensor’ function with Periodicity of 5 ms and Priority of 1.



IV . Sending Data:

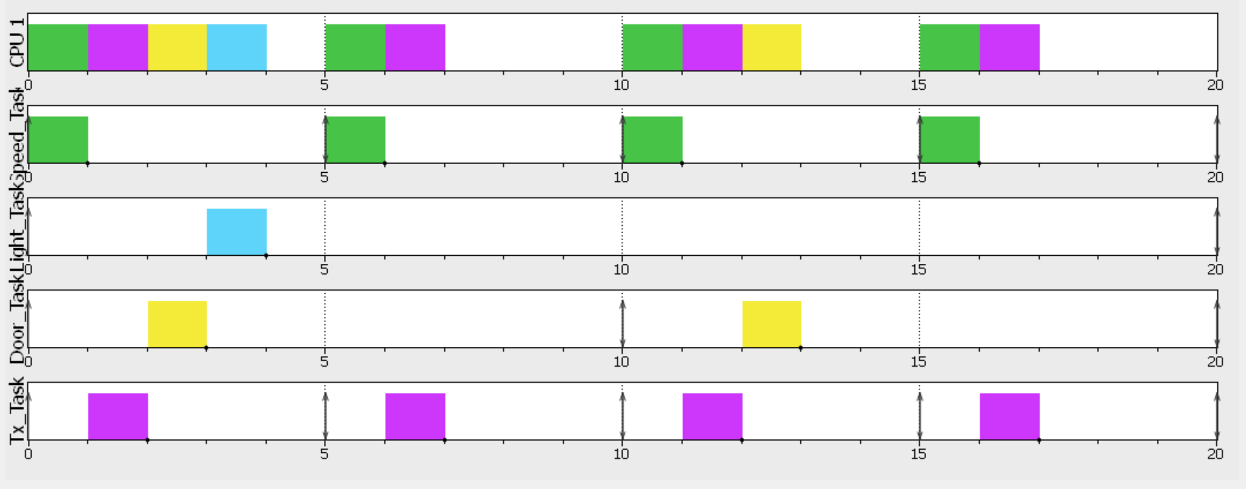
After taking sensors’ and switch readings, we will need a task to create data frame and send data through CAN Bus, that what Tx_Task do.



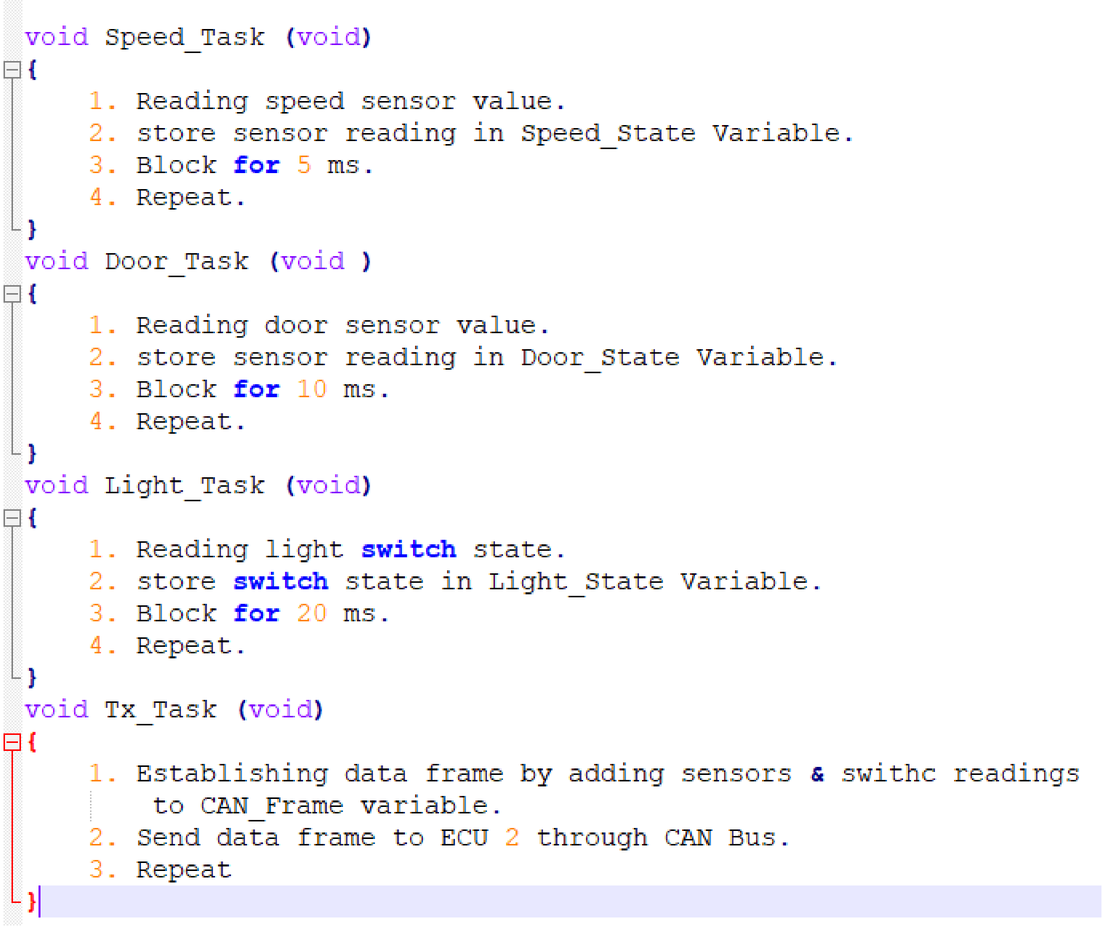
Its Periodicity is 5 ms, but it will have the least priority in the system, because we need to wait other tasks to be executed to form data frame. So, let it Priority to be 2.

V. Simso & Pesudo Code:

After knowing priority of all tasks & tasks' periodicities, we can use simso to have a better look to our system (assuming execution time of all tasks is 1 ms)



Pesudo Code of ECU 1:



```
void Speed_Task (void)
{
    1. Reading speed sensor value.
    2. store sensor reading in Speed_State Variable.
    3. Block for 5 ms.
    4. Repeat.
}
void Door_Task (void )
{
    1. Reading door sensor value.
    2. store sensor reading in Door_State Variable.
    3. Block for 10 ms.
    4. Repeat.
}
void Light_Task (void)
{
    1. Reading light switch state.
    2. store switch state in Light_State Variable.
    3. Block for 20 ms.
    4. Repeat.
}
void Tx_Task (void)
{
    1. Establishing data frame by adding sensors & swithc readings
       to CAN_Frame variable.
    2. Send data frame to ECU 2 through CAN Bus.
    3. Repeat
}
```

C. Modules' APIs Description & Typedef Description:

I. Application Layer:

1. **void ValUpdate(uint32 *NewVal, uint32 NewData)**

- **Function Description:** Updating variables with new data.

- **Arguments:** 1. NewVal: Pointer to uint32 which holds address of the variable which need to be updated.

2. NewData: New data with which the variable will be updated.

` - **Return Description:** void

- Typedefs Description: uint32: unsigned 32 bit integer value.

II. Services Layer:

1. void ComManager_Tx (uint8 protocol, uint32 *ptrData)

- **Module:** Com Manager.

- Function Description: Transmit / Send data through communication protocols.

- Arguments: 1. Protocol: Communication protocol to send data with.

2. ptrData: Pointer to data which be sent.

- **Return Description:** void

- **Typedefs Description:** 1. uint8: unsigned 8 bit integer value.

2. uint32 : unsigned 32 bit integer value.

III. On-Board Layer:

1. void Init_Door_Sensor (DoorCfgStr *DoorPtr)

- **Module:** Door_Sensor.

- Function Description: Initializing GPIO Pins for door sensor

- Arguments: DoorPtr: Pointer to DoorCfgStr structure.

- **Return Description:** void

- **Typedefs Description:** DoorCfgStr: structure used to initialize pins for door sensor (as it holds configurations for initializing pins).

2. **DoorType DoorState(DoorCfgStr *SensorPtr)**

- **Module:** Door_Sensor.

- **Function Description:** Function returns HIGH or LOW (Depending on door state)

- **Arguments:** SensorPtr: Pointer to DoorCfgStr structure which referes to required door sensor.

- **Return Description:** DoorType: Enum wih HIGH, LOW

3. **void Init_Speed_Sensor (SpeedCfgStr *Speedptr)**

- **Module:** Speed_Sensor.

- **Function Description:** Initializing GPIO Pins for speed sensor.

- **Arguments:** Speedptr: Pointer to SpeedCfgStr structure.

- **Return Description:** void

- **Typedefs Description:** SpeedCfgStr: structure used to initialize pins for speed sensor (as it holds configurations for initializing pins).

4. **float SpeedState(SpeedCfgStr *Sptr)**

- **Module:** Speed_Sensor.

- **Function Description:** Function returns values (float) from speed sensor.

- **Arguments: Sptr**: Pointer to SpeedCfgStr structure which referes to required speed sensor.

- **Return Description:** float speed value.

- **Typedefs Description:** SpeedCfgStr: structure used to initialize pins for speed sensor (as it holds configurations for initializing pins).

5. void Init_Switch (SwitchCfgStr *SwPtr)

- **Module:** Light_Switch.

- **Function Description:** Initializing GPIO Pins for Light Switch.

- **Arguments: SwPtr**: Pointer to SwitchCfgStr structure.

- **Return Description:** void

- **Typedefs Description:** SwitchCfgStr: structure used to initialize pins for light switch (as it holds configurations for initializing pins).

6. **SWType SWState(** SwitchCfgStr ***Switchptr)**

- **Module:** Light_Switch.

- **Function Description:** Function returns HIGH or LOW (Depending on switch state)

- **Arguments:** Switchptr: Pointer to SwitchCfgStr structure which referes to required light Switch.

- **Return Description:** SWType: Enum wih HIGH, LOW

IV. MCAL:

1. **void Init_Timer(TimerCfgStr *Tptr)**

- **Module:** Timer.

- **Function Description:** Initializing timer with required operation.

- **Arguments:** Tptr: Pointer to TimerCfgStr structure.

- **Return Description:** void

- **Typedefs Description:** TimerCfgSt: structure used to initialize timer (as it holds configurations for timer operations).

2. **void TimerStart (TType)**

- **Module:** Timer.

- **Function Description:** Function which makes timer start counting.

- **Arguments:** Indicates which timer to be used

- **Return Description:** void

3. **void TimerStop (TType)**

- **Module:** Timer.

- **Function Description:** Function which makes timer stop counting.

- **Arguments:** Indicates which timer to be used

- **Return Description:** void

4. **void Delay_Ms (ms)**

- **Module:** Timer.

- **Function Description:** Delay function.

- **Arguments:** Value represents delay in ms.

- **Return Description:** void

5. void Init_DIO(DIOCfgStr *DIOptr)

- **Module:** DIO.

- **Function Description:** Initializing GPIO pins as DIO.

- **Arguments:** DIOptr: Pointer to DIOCfgStr structure.

- **Return Description:** void

- **Typedefs Description:** DIOCfgStr: structure used to initialize DIO pins (as it holds configurations for GPIO pins).

6. void DIO_Write(PinType Pin, PortType Port, ValueType Val)

- **Module:** DIO.

- **Function Description:** Function that writes (HIGH or LOW) in specific pins determine by user.

- **Arguments:** Indicates which port, pin and value you want to write on.

- **Return Description:** void

7. ValType DIO_Read(PinType pin, PortType port)

- **Module:** DIO.

- **Function Description:** Function that reads state of pin.

- **Arguments:** Indicates which port and pin you want to read.

- **Return Description:** ValType: Enum with state HIGH, LOW.

8. void DIO_Tgl (Pin_Type pin, Port_Type port)

- **Module:** DIO.

- **Function Description:** Function that toggles a specific pin determined by user.

- **Arguments:** Indicates which port and pin you want to toggle.

- **Return Description:** void

9. void Init_ADC(DIOCfgStr *ADCptr)

- **Module:** ADC.

- **Function Description:** Initializing GPIO pins as ADC.

- **Arguments** ADCptr: Pointer to DIOCfgStr structure.

- **Return Description:** void

- **Typedefs Description:** DIOCfgStr: structure used to initialize DIO pins (as it holds configurations for GPIO pins).

10. float ADC_Read(PinType pin, PortType port)

- **Module:** ADC.

- Function Description: Function that reads analog values from pins determined by user.

- Arguments: Indicates which port and pin you want to read from.

- **Return Description:** float (as it an analog value)

11 . void Init_CAN(DIOCfgStr *CANptr)

- **Module:** CAN.

- Function Description: Initializing GPIO pins as CAN.

- Arguments CANptr: Pointer to DIOCfgStr structure.

- **Return Description:** void

- **Typedefs Description:** DIOCfgStr: structure used to initialize DIO pins (as it holds configurations for GPIO pins).

12 . void CAN_Tx(uint32_t *Sdata**)**

- **Module:** CAN.

- Function Description: Sending data through CAN Bus.

- Arguments Sdata: Pointer to data that will be sent.

- **Return Description:** void

12 . void CAN_Rx(uint32_t *Rdata**)**

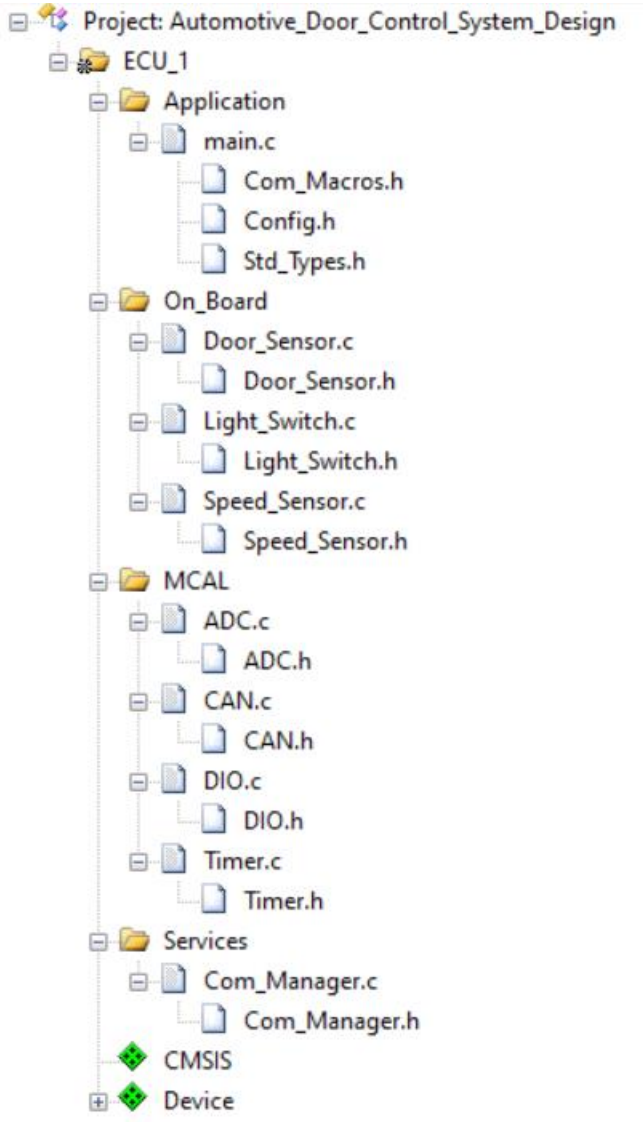
- **Module:** CAN.

- Function Description: Receiving data from CAN Bus.

- Arguments Rdata: Pointer that receiving data (storing data).

- **Return Description:** void

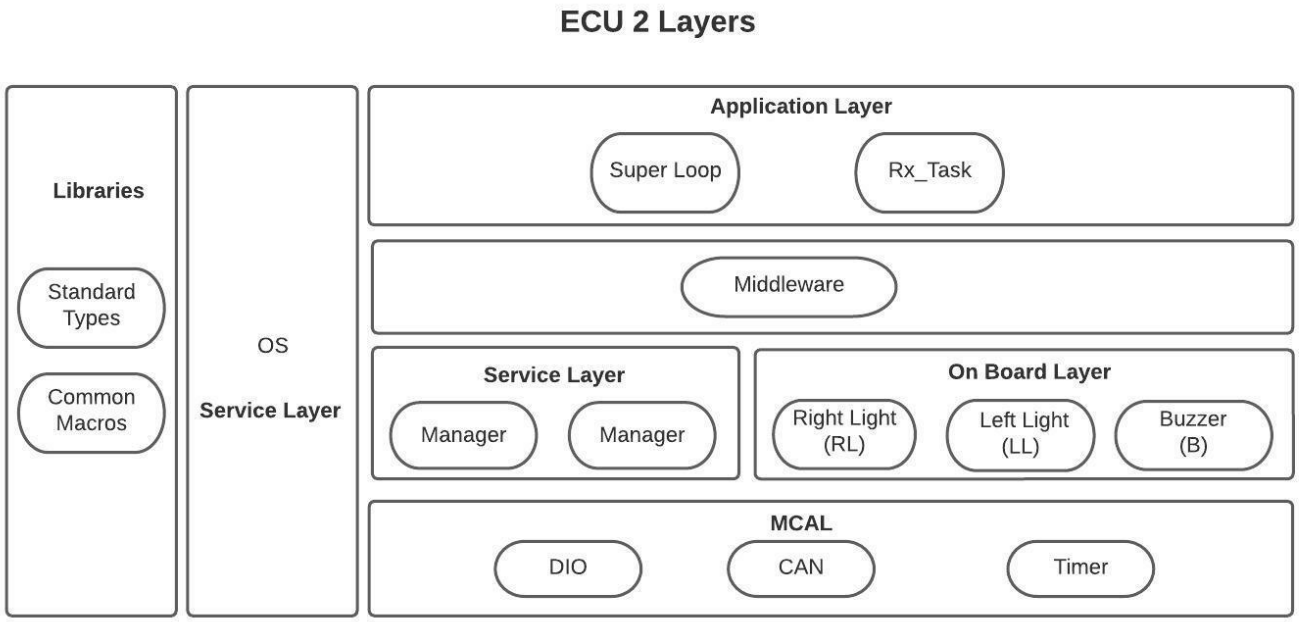
d. Folder Structure:



```
Project: Automotive_Door_Control_System_Design
    ECU_1
        Application
            main.c
                Com_Macros.h
                Config.h
                Std_Types.h
        On_Board
            Door_Sensor.c
                Door_Sensor.h
            Light_Switch.c
                Light_Switch.h
            Speed_Sensor.c
                Speed_Sensor.h
        MCAL
            ADC.c
                ADC.h
            CAN.c
                CAN.h
            DIO.c
                DIO.h
            Timer.c
                Timer.h
        Services
            Com_Manager.c
                Com_Manager.h
        CMSIS
        Device
```

III. ECU 2:

a. Layered architecture:

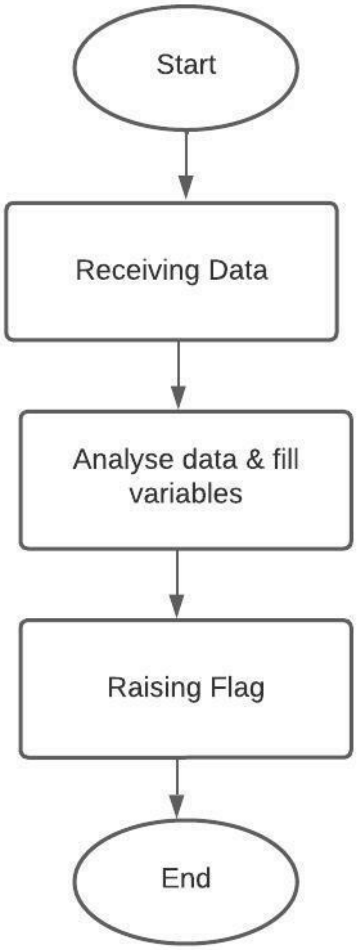


b. Components And Modules:

We'll use Event triggered OS with ECU 2, as it triggered by receiving sensors & switch states from ECU 1 through CAN Bus, then taking actions based on received values.

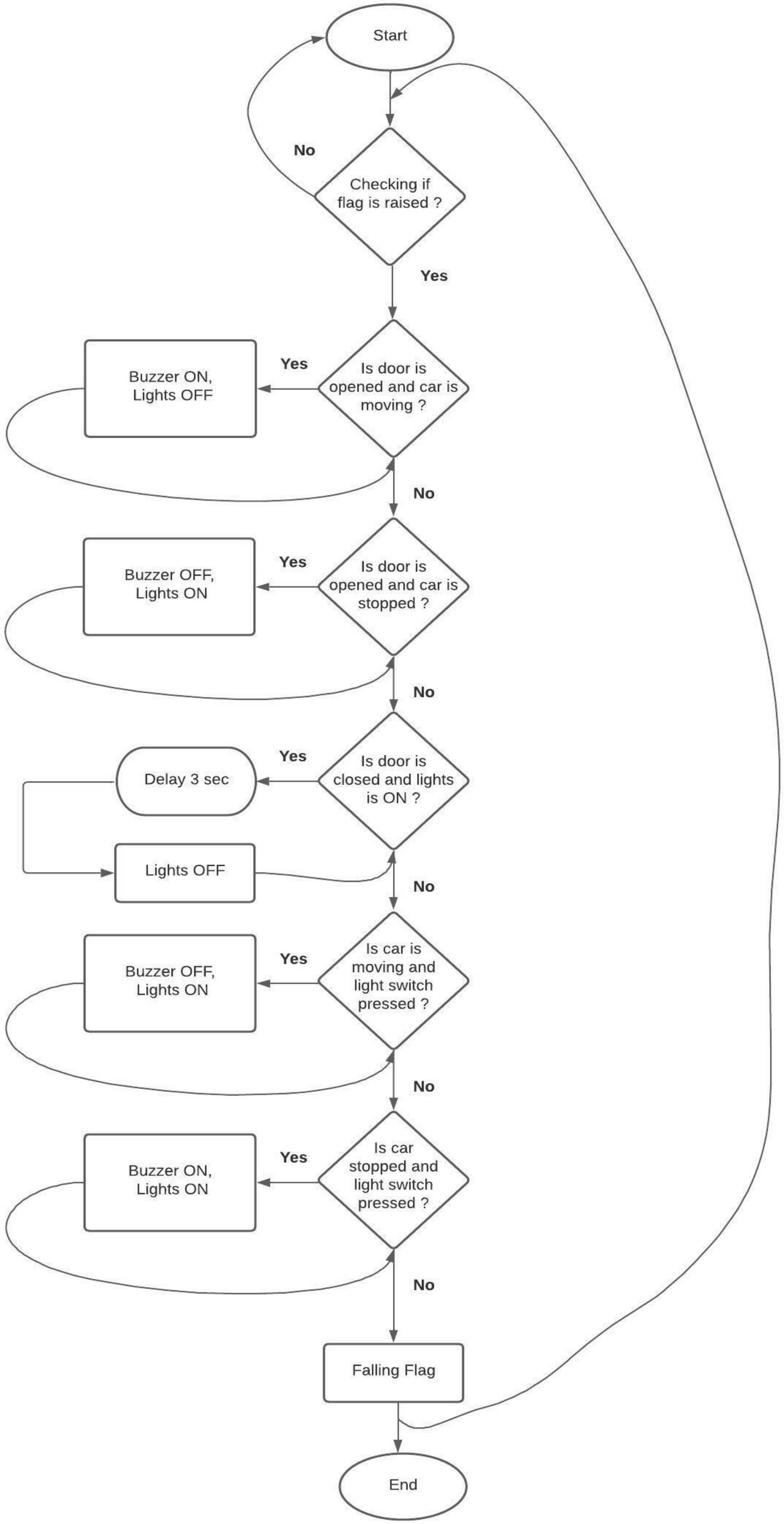
i. Rx_Task

Rx_Task is triggered by CAN Interrupt, its function is to receive data frame from CAN Bus, updating variables with new data, then raising the flag, so Control_Task can run



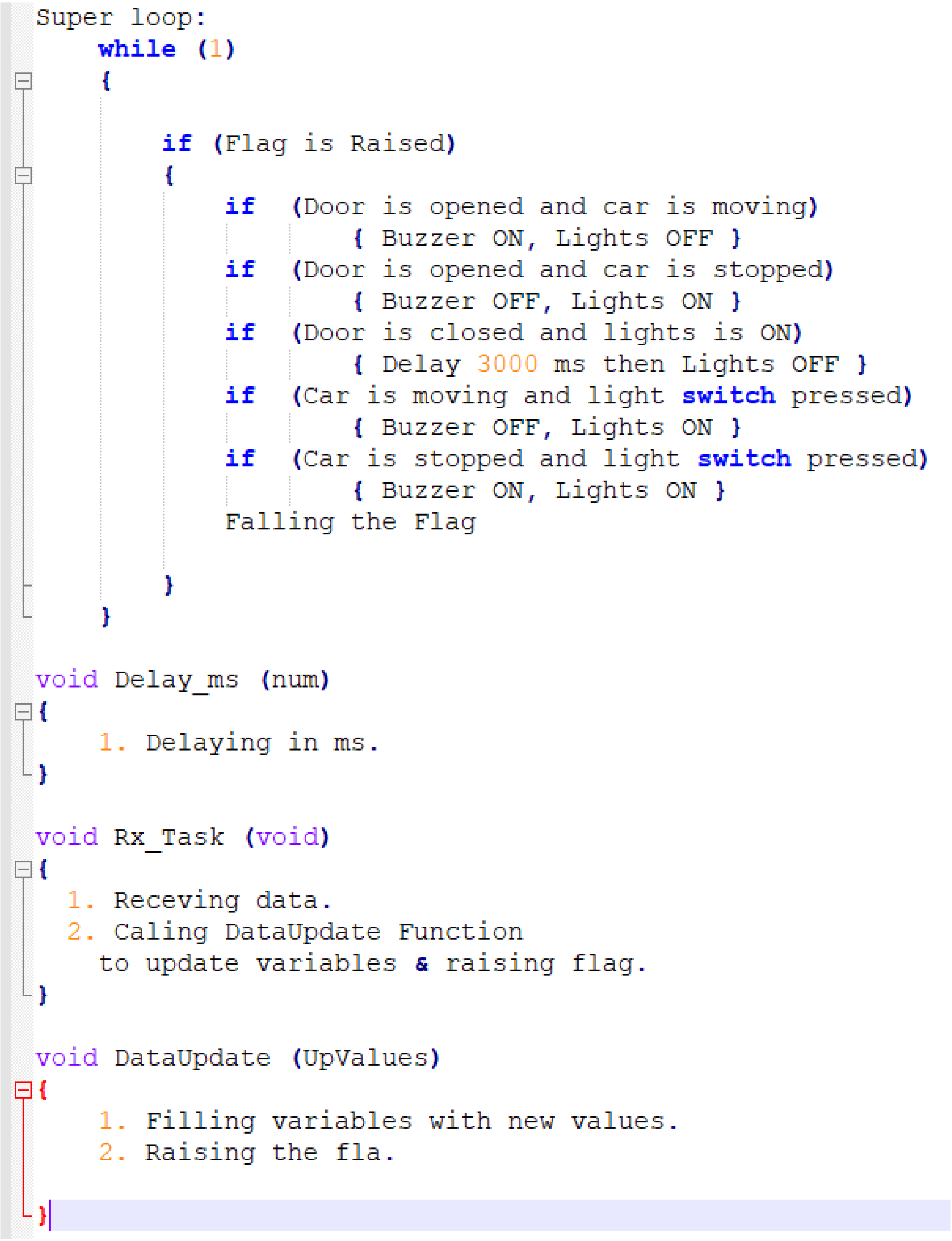
ii. Control_Task

Its controls RL, LL and Buzzer by making actions based on data received from CAN Bus (as described in flowchart).



Start
No
Checking if flag is raised ?
Yes
Yes
Buzzer ON, Lights OFF
Is door is opened and car is moving ?
No
Yes
Buzzer OFF, Lights ON
Is door is opened and car is stopped ?
No
Yes
Delay 3 sec
Is door is closed and lights is ON ?
Lights OFF
No
Yes
Buzzer OFF, Lights ON
Is car is moving and light switch pressed ?
No
Yes
Buzzer ON, Lights ON
Is car stopped and light switch pressed ?
No
Falling Flag
End

Pesudo Code of ECU 2:



```
Super loop:
    while (1)
    {

        if (Flag is Raised)
        {
            if  (Door is opened and car is moving)
                    { Buzzer ON, Lights OFF }
            if  (Door is opened and car is stopped)
                    { Buzzer OFF, Lights ON }
            if  (Door is closed and lights is ON)
                    { Delay 3000 ms then Lights OFF }
            if  (Car is moving and light switch pressed)
                    { Buzzer OFF, Lights ON }
            if  (Car is stopped and light switch pressed)
                    { Buzzer ON, Lights ON }
            Falling the Flag

        }
    }

void Delay_ms (num)
{
    1. Delaying in ms.
}

void Rx_Task (void)
{
  1. Receving data.
  2. Caling DataUpdate Function
    to update variables & raising flag.
}

void DataUpdate (UpValues)
{
    1. Filling variables with new values.
    2. Raising the fla.

}
```

C. Modules' APIs Description & Typedef Description:

I. Application Layer:

1. **Void ValUpdate(uint32 *DUpdate, uint32 *LUpdate, uint32 *SUpdate)**

- **Function Description:** Updating variables with new data.

- **Arguments:** 1. **DUpdate** Pointer to uint32 which holds address of the new data received from door sensor.

2. **LUpdate**: Pointer to uint32 which holds address of the new data received from Light switch.

3. **SUpdate:** Pointer to uint32 which holds address of the new data received from speed sensor.

` - **Return Description:** void

- Typedefs Description: uint32: unsigned 32 bit integer value.

II. In-Board Layer:

1. void Init_Buzzer (BuzzerCfgStr *Bptr)

- **Module:** Buzzer.

- Function Description: Initializing GPIO Pins for the buzzer.

- Arguments: Bptr: Pointer to DoorCfgStr structure.

- **Return Description:** void

- **Typedefs Description:** BuzzerCfgStr: structure used to initialize pins for the buzzer (as it holds configurations for initializing pins).

2. void BuzzerState (BuzzerCfgSt *Bptr, SType st)

- **Module:** Buzzer.

- Function Description: Setting buzzer state (Active or Inactive).

- Arguments: Bptr: Pointer to BuzzerCfgSt structure referrers to used buzzer.

- **Return Description:** void

- **Typedefs Description:** 1. BuzzerCfgStr: structure used to initialize pins for the buzzer (as it holds configurations for initializing pins).

2. SType: State (Active or Inactive)

3. Void Init_Light (LightCfgStr *Lptr)

- **Module:** Lights.

- **Function Description:** Initializing GPIO Pins for RL & LL.

- **Arguments:** Lptr: Pointer to LightCfgStr structure.

- **Return Description:** void

- **Typedefs Description:** LightCfgStr: structure used to initialize pins for LL & RL (as it holds configurations for initializing pins).

2. void LightState(LightCfgStr *Lptr, SType st)

- **Module:** Lights.

- **Function Description:** Setting lights state (ON or OFF).

- **Arguments: Lptr:** Pointer to LightCfgStr structure that refers to required light**.**

- **Return Description:** void

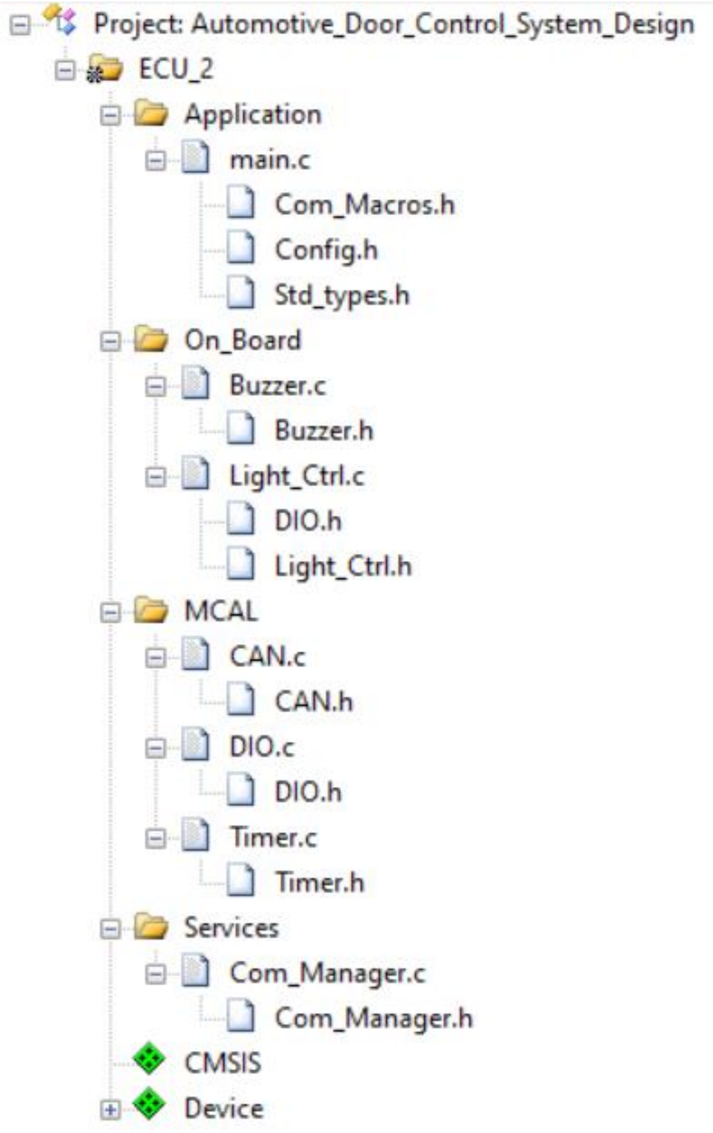
- **Typedefs Description:** 1. LightCfgStr : structure used to initialize pins for the buzzer (as it holds configurations for initializing pins).

2. SType: State (ON or OFF).

III. Services Layer: (Same as ECU 1)

IV. MCAL Layer: (Same as ECU 1) but without ADC module

d. Folder Structure:



```
Project: Automotive_Door_Control_System_Design
    ECU_2
        Application
            main.c
                Com_Macros.h
                Config.h
                Std_types.h
        On_Board
            Buzzer.c
                Buzzer.h
            Light_Ctrl.c
                DIO.h
                Light_Ctrl.h
        MCAL
            CAN.c
                CAN.h
            DIO.c
                DIO.h
            Timer.c
                Timer.h
        Services
            Com_Manager.c
                Com_Manager.h
        CMSIS
        Device
```