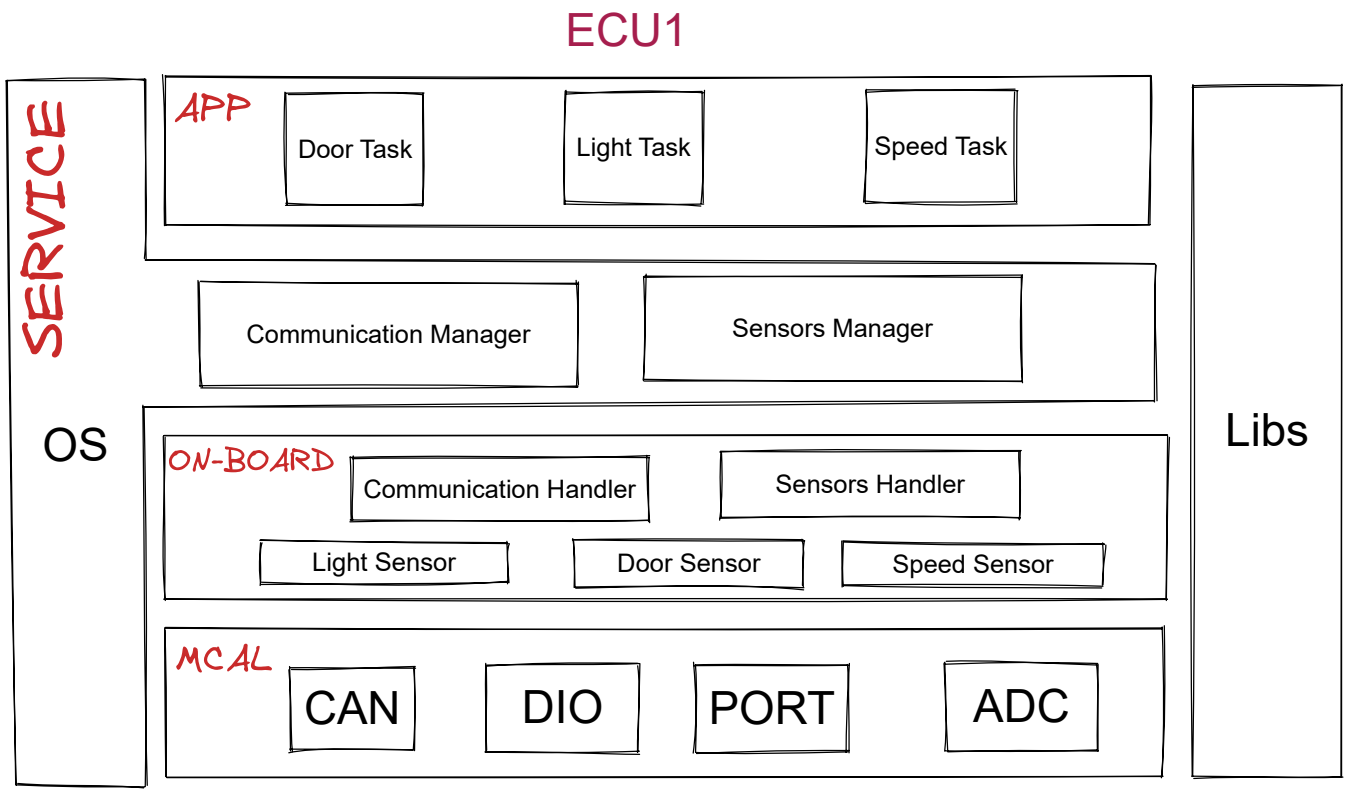
# Static Design

## ECU1

### Layered Architecture and Modules.




- since I'm not sending or receiving data from any external hardware component that will be in the on-board layer, I can neglect the communication handler in this case
- Also, I only have on communication manager so there is no need to add the middleware layer
- Note: Vehicles use Hall effect sensors to determine speed. All a Hall effect sensor does is detect the presence of a magnetic field. So a magnet is mounted to whatever is spinning and the H.E. sensor is mounted to a stationary point. Every time the wheel makes a makes a revolution the magnet passes by the H.E. sensor and then the microprocessor does the math to determine the RPM which is then translated into MPH.

---

**CAN**

---

| Name | Can_Init |
|---|---|
| Syntax | `void Can_Init(const Can_ConfigType *config)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | Config |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | Initialize the module |

| Name | Can_DeInit |
|---|---|
| Syntax | `void Can_DeInit(void)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | None |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | Deinitialize the module |

| Name | Can_Send |
|---|---|
| Syntax | `void Can_Send(uint32 msg)` |
| Sync/Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | msg |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | sends data through the CAN peripheral |

- Typedefs

| Name | Can_ConfigType |
| --- | --- |
| Type | Struct |
| Description | A struct that holds all of the peripheral initialization |

---

# Port

| Name | Port_Init |
| --- | --- |
| Syntax | `void Port_Init(Port_ConfigType *config)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | config |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | Initialize the module |

| Name | Port_DeInit |
| --- | --- |
| Syntax | `void Port_DeInit(void)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | None |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | Deinitialize the module |

- Typedefs

| Name | Port_ConfigType |
| --- | --- |
| Type | Struct |
| Description | A struct that holds all of the peripheral initialization |

# DIO

| Name | Dio_ReadChannel |
|---|---|
| Syntax | `Dio_LevelType Dio_ReadChannel(Dio_ChannelType channelId)` |
| Sync/Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | channelId |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | Dio_LevelType |
| Description | Read a specific MCU pin state |

| Name | Dio_WriteChannel |
|---|---|
| Syntax | `void Dio_WriteChannel(Dio_ChannelType channelId, Dio_LevelType level)` |
| Sync/Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | channelId, level |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | Set a specific pin level either to high or low |

- Typedefs

| Name | Dio_ChannelType |
|---|---|
| Type | uint8 |
| Description | This type identifies which pin on the MCU we need to write to or read from |

| Name | Dio_LevelType |
|---|---|

| Name | Dio_LevelType |
| --- | --- |
| Type | uint8 |
| Description | This type identifies The state of the pin either high or low |

---

# ADC

| Name | Adc_Init |
| --- | --- |
| Syntax | `void Adc_Init(void)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | None |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | Initialize the module |

| Name | Adc_DeInit |
| --- | --- |
| Syntax | `void Adc_DeInit(void)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | None |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | Deinitialize the module |

| Name | Adc_ReadData |
| --- | --- |
| Syntax | `u16 Adc_ReadData(u8 channel)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |

---

| Name | Adc_ReadData |
|---|---|
| Parameters (in) | channel |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | u16 |
| Description | Get the current reading from the ADC |

---

## Speed Sensor

| Name | Ss_GetValue |
|---|---|
| Syntax | `u16 Ss_GetValue(void)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | None |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | u16 |
| Description | Gets the current speed value using the Hall Effect Sensor |

---

## Door Sensor

| Name | Ds_GetDoorState |
|---|---|
| Syntax | `Ds_StateType Ds_GetDoorState(void)` |
| Sync/Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | None |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | Ds_StateType |

---

| Name | Ds_GetDoorState |
| --- | --- |
| Description | gets the current state of the door |

- Typedefs

| Name | Ds_StateType |
| --- | --- |
| Type | enum |
| Description | This type identifies The state of the door either OPEN or CLOSED |

---

## Light Sensor

| Name | Ls_GetLightState |
| --- | --- |
| Syntax | `Ls_StateType Ls_GetLightState(void)` |
| Sync/Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | None |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | Ls_StateType |
| Description | Gets the current state of the light switch |

- Typedefs

| Name | Ls_StateType |
| --- | --- |
| Type | enum |
| Description | This type identifies The state of the light switch either ON or OFF |

---

## Sensors Handler

| Name | Sh_Select |
| --- | --- |
| Syntax | `u32 Sh_Select(u8 Id)` |

---

| Name | Sh_Select |
|---|---|
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | Id |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | u32 |
| Description | Chooses which sensor to read the data from |

---

## Communication Handler

| Name | Ch_Send |
|---|---|
| Syntax | `void Ch_Send(u32 msg, u8 Id)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | msg, Id |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | Chooses which bus or peripheral through which it sends the data |

---

## Sensors Manger

| Name | Sm_Select |
|---|---|
| Syntax | `u32 Sm_select(u8 Id)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | Id |

| Name | Sm_Select |
| --- | --- |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | u32 |
| Description | An interface for the application layer to Chooses which sensor to read the data from |

---

# Communication Manger

| Name | Cm_Send |
| --- | --- |
| Syntax | `void Cm_Send(u8 Id, u32 msg)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | Id, msg |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | An interface for the application layer to Chooses which bus or peripheral through which it sends the data |

---

# Door Task

1. APIs

| Name | task_SendDoorState |
| --- | --- |
| Syntax | `void task_SendDoorState(void)` |

---

| Name | task_SendDoorState |
| --- | --- |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | None |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | A task that sends the door state through the can protocol to another ECU every 10ms |

---

# Light Task

1. APIs

| Name | task_SendLightSwitchState |
| --- | --- |
| Syntax | `void task_SendLightSwitchState(void)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | None |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | A task that sends the light switch state through the can protocol to another ECU every 20ms |

---

# Speed Task

1. APIs

| Name | task_SendSpeedState |
|---|---|
| Syntax | `void task_SendSpeedState(void)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | None |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | A task that sends the speed state through the can protocol to another ECU every 5ms |

---

## Folder Structure

- **App/**
  - main.c
- **Libs/**
  - std_types.h
  - bit_math.h
- **Service/**
  - Include/
    - OS.h
    - Com_Mgr.h
    - Sens_Mgr.h
  - OS.c
  - Com_Mgr.c
  - Sens_Mgr.c
- **Hal/**
  - Include/
    - Com_Handler.h
    - Sens_Handler.h
    - Light_Sensor.h
    - Speed_Sensor.h

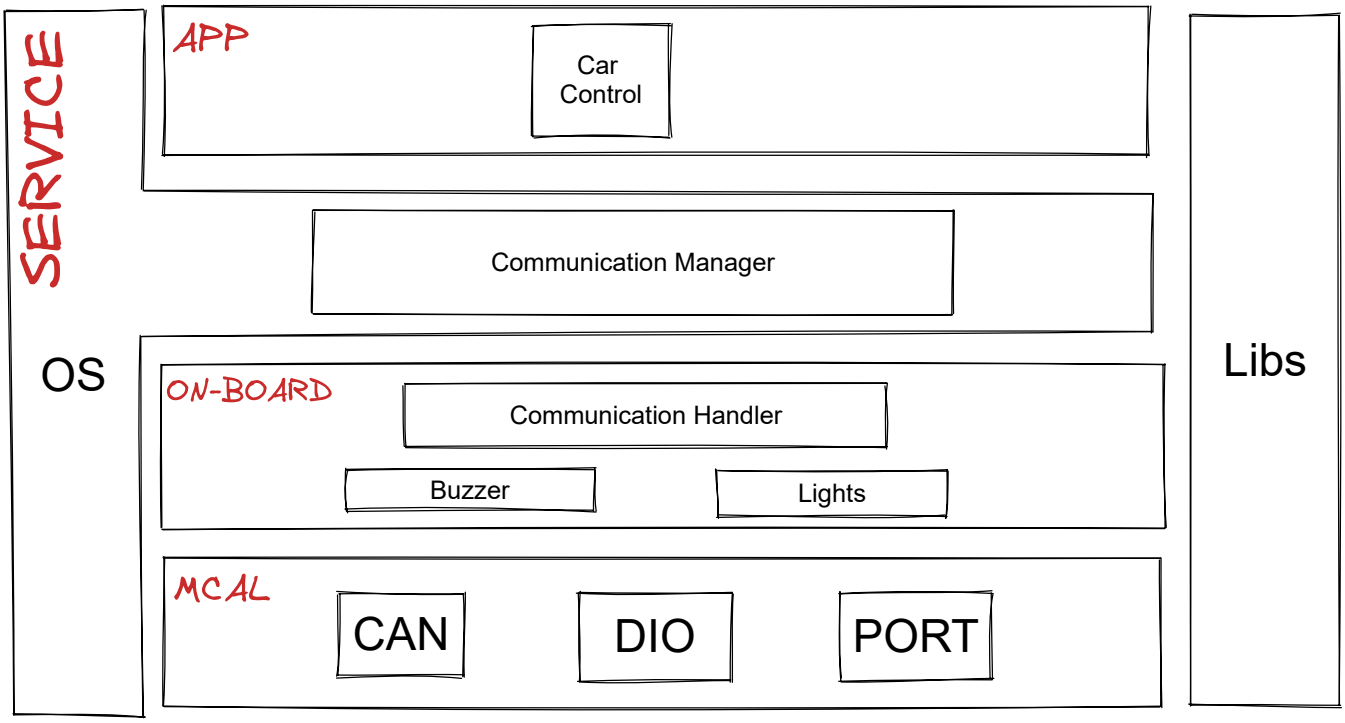
        - Door_Sensor.h
    - Com_Handler.c
    - Sens_Handler.c
    - Light_Sensor.c
    - Speed_Sensor.c
    - Door_Sensor.c
- Mcal/
    - CAN/
        - Config/
            - Can_Cfg.h
            - Can_Lcfg.c
        - Can.h
        - Can.c
    - DIO/
        - DIO.h
        - DIO.c
    - PORT/
        - Config/
            - Port_Cfg.h
            - Port_Lcfg.c
        - Port.h
        - Port.c
    - ADC/
        - Config/
            - Adc_Cfg.h
            - Adc_Lcfg.c
        - Adc.h
        - Adc.c

---

# ECU2

## Layered Architecture and Modules.

# ECU2




- since I'm not sending or receiving data from any external hardware component that will be in the on-board layer, I can neglect the communication handler in this case
- Also, I only have on communication manager so there is no need to add the middleware layer
- Note: Vehicles use Hall effect sensors to determine speed. All a Hall effect sensor does is detect the presence of a magnetic field. So a magnet is mounted to whatever is spinning and the H.E. sensor is mounted to a stationary point. Every time the wheel makes a makes a revolution the magnet passes by the H.E. sensor and then the microprocessor does the math to determine the RPM which is then translated into MPH.

---

## CAN

| Name | Can_Init |
|---|---|
| Syntax | `void Can_Init(const Can_ConfigType *config)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | Config |
| Parameters (inout) | None |
| Parameters (out) | None |

| Name | Can_Init |
| --- | --- |
| Return Value | None |
| Description | Initialize the module |

| Name | Can_DeInit |
| --- | --- |
| Syntax | `void Can_DeInit(void)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | None |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | Deinitialize the module |

| Name | Can_Read |
| --- | --- |
| Syntax | `uint32 Can_Read(void)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | None |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | uint32 |
| Description | reads data through the CAN peripheral |

- Typedefs

| Name | Can_ConfigType |
| --- | --- |
| Type | Struct |
| Description | A struct that holds all of the peripheral initialization |

---

# Port

| Name | Port_Init |
|---|---|
| Syntax | `void Port_Init(Port_ConfigType *config)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | config |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | Initialize the module |

| Name | Port_DeInit |
|---|---|
| Syntax | `void Port_DeInit(void)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | None |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | Deinitialize the module |

- Typedefs

| Name | Port_ConfigType |
|---|---|
| Type | Struct |
| Description | A struct that holds all of the peripheral initialization |

---

# DIO

| Name | Dio_ReadChannel |
|---|---|

| Name | Dio_ReadChannel |
|---|---|
| Syntax | `Dio_LevelType Dio_ReadChannel(Dio_ChannelType channelId)` |
| Sync/Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | channelId |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | Dio_LevelType |
| Description | Read a specific MCU pin state |

| Name | Dio_WriteChannel |
|---|---|
| Syntax | `void Dio_WriteChannel(Dio_ChannelType channelId, Dio_LevelType level)` |
| Sync/Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | channelId, level |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | Set a specific pin level either to high or low |

- Typedefs

| Name | Dio_ChannelType |
|---|---|
| Type | uint8 |
| Description | This type identifies which pin on the MCU we need to write to or read from |

| Name | Dio_LevelType |
|---|---|
| Type | uint8 |
| Description | This type identifies The state of the pin either high or low |

## Lights

| Name | lights_SetState |
|---|---|
| Syntax | `void lights_SetState(u8 Id, u8 level)` |
| Sync/Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | Id, level |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | It sets the light determined by the provided ID to either on or off |

## Buzzer

| Name | buzzer_SetState |
|---|---|
| Syntax | `void buzzer_SetState(u8 level)` |
| Sync/Async | Synchronous |
| Reentrancy | Reentrant |
| Parameters (in) | level |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | It sets the buzzer to either on or off |

## Communication Handler

| Name | Ch_Send |
|---|---|
| Syntax | `void Ch_Send(u32 msg, u8 Id)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |

| Name | Ch_Send |
| --- | --- |
| Parameters (in) | msg, Id |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | Chooses which bus or peripheral through which it sends the data |

---

## Communication Manger

| Name | Cm_Send |
| --- | --- |
| Syntax | `void Cm_Send(u8 Id, u32 msg)` |
| Sync/Async | Synchronous |
| Reentrancy | Non Reentrant |
| Parameters (in) | Id, msg |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | An interface for the application layer to Chooses which bus or peripheral through which it sends the data |

---

## Car Control

1. APIs

| Name | carCtrl_UpdateState |
| --- | --- |
| Syntax | `void carCtrl_UpdateState(void)` |
| Sync/Async | Synchronous |

---

| Name | carCtrl_UpdateState |
| --- | --- |
| Reentrancy | Non Reentrant |
| Parameters (in) | None |
| Parameters (inout) | None |
| Parameters (out) | None |
| Return Value | None |
| Description | A function That updates the internal car control module state with the newly received state from ECU1 and acts on the lights and the buzzer accordingly. |

---

## Folder Structure

- `App/`
  - main.c
- `Libs/`
  - std_types.h
  - bit_math.h
- `Service/`
  - `Include/`
    - OS.h
    - Com_Mgr.h
  - OS.c
  - Com_Mgr.c
- `Hal/`
  - `Include/`
    - Com_Handler.h
    - Lights.h
    - Buzzer.h
  - Com_Handler.c
  - Lights.c
  - Buzzer.c
- `Mcal/`

- CAN/
  - Config/
    - Can_Cfg.h
    - Can_Lcfg.c
  - Can.h
  - Can.c
- DIO/
  - DIO.h
  - DIO.c
- PORT/
  - Config/
    - Port_Cfg.h
    - Port_Lcfg.c
  - Port.h
  - Port.c