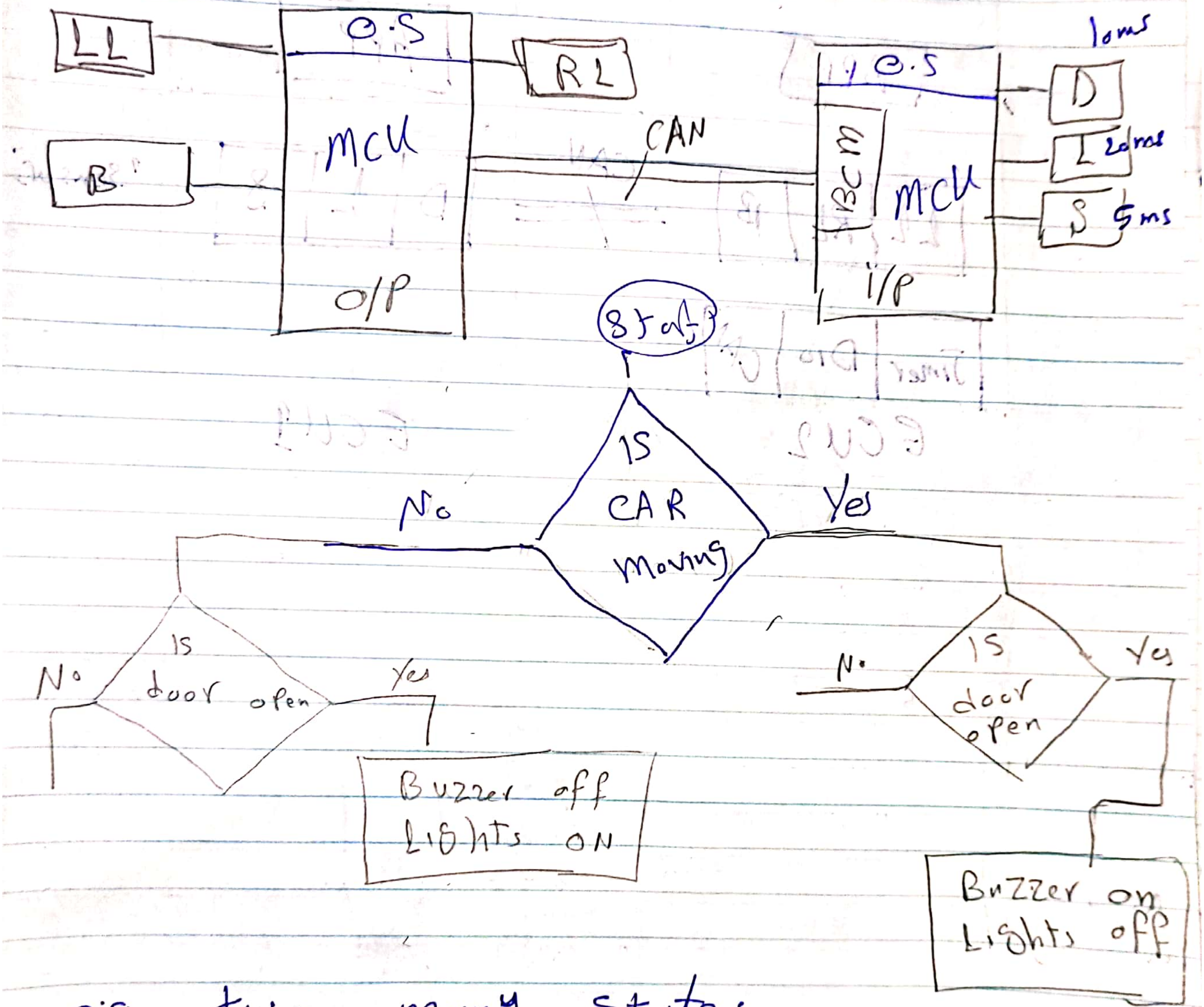
Ahmed M. Shindy

Egfwd : Static Design

" Automotive Door Control System Design "

" Block diagram "



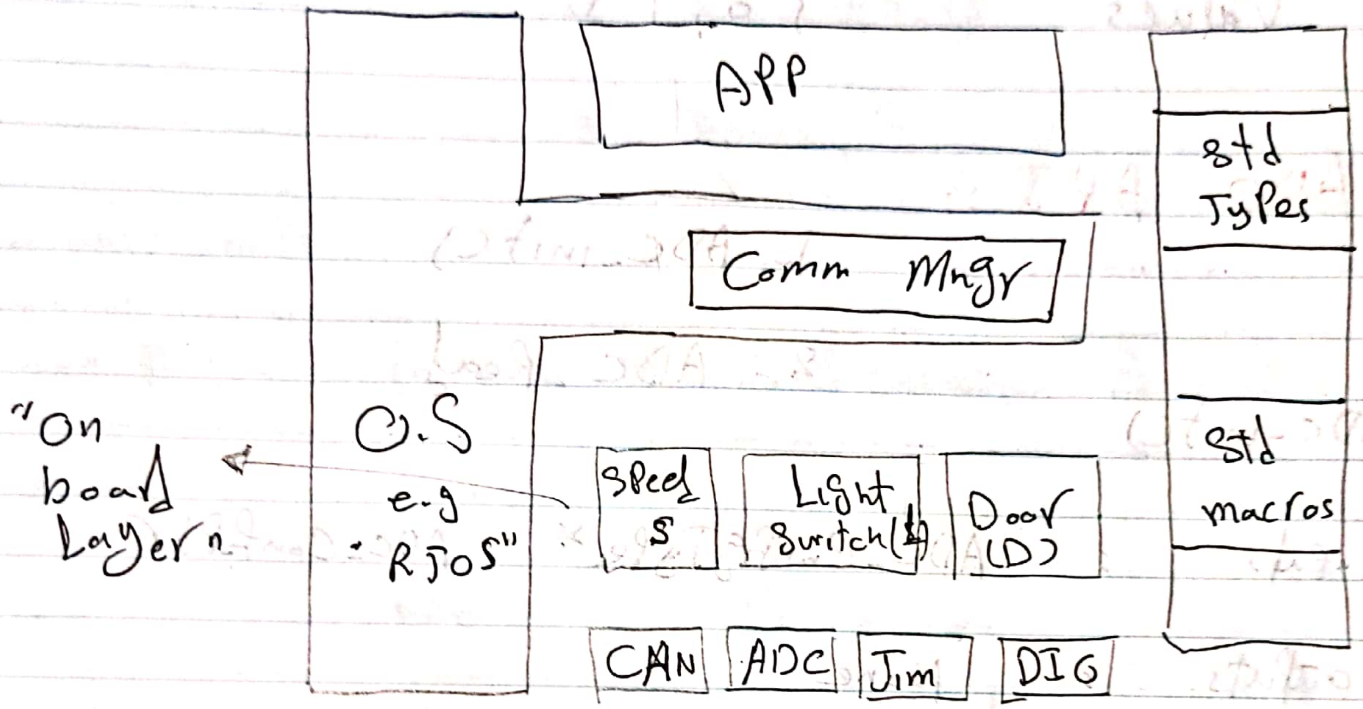
∴ two many states

↳ bad approach " Flow chart "

, No need to complicate things

NOTE:

For ECU's : Static Design



"ECU's" Layers

Each Module API :

I. DIO 1- Void Dio_init (Port, Pin, level);

2 Void Dio_write(Port, ~ , ~ );

3 _ uint8 Dio_Read (Port, ~);

_ Port_t enum or macro {A ~> F) ~~or more~~

- Dio_Channel {chA → chx} enum or macro.

| | |
|---|---|
| Dio_PinLevel | Type in C ; enum/macro |
| Values | {0, 1} |

## 2- ADC API(s)

1- ADC_init()

2- ADC_Read

1- ADC_init()

| | | |
|---|---|---|
| inputs | : | ADC_ConfigType* ADC_ConfPtr |
| outputs | : | None |
| Return | : | True or False |
| Function | : | initializers ADC module |

2- ADC_Read()

| | | |
|---|---|---|
| input | : | Port Number , Channel Number |
| output | : | None |
| Return | : | uint16 or uint32 not ~~uint8~~ |
| functions | : | Reads analog value of Sensor. |

## Timer API(s) :

1- Tim_Init()

2- Timer_Start()

3- Timer_Stop()

### 1- TimerInit()

| | |
|---|---|
| input : | Pointer to Configuration Struct |
| output : | None |
| Return : | ~~None~~ True/ false |
| function : | Initializes Timer Module |

### 2- Timer_Start() & Timer_Stop()

| | |
|---|---|
| input : | None |
| output : | None |
| Return : | True or false |
| function : | Start / Stop Timer |

## CAN API(s):

1- CAN_Init()

2- CAN_Transmit ( )

3- CAN_Recieve ( )

### 1-CAN_Init()

| | |
|---|---|
| Inputs : | Pointer to configuration struct |
| outputs : | No |
| Return | True/False |

### 2- CAN_Transmit()

| | |
|---|---|
| Input : | Data with size x "uint32" |
| output : | None |
| Return : | 1/0 |
| function : | Transmits CAN frame |

### 3. CAN_Recieve( )

the Same as CAN_Tx but Returns data with Size (x)

② Door Sensor API(s)　　1- Door-Sen-Init()
　　　　　　　　　　　2- 〃 〃 Read-State()

1- DoorSen-Init()

Input : None

output : None

~~Return : closed(0) or open(1) bool value~~

2- DoorSen-Read_State()

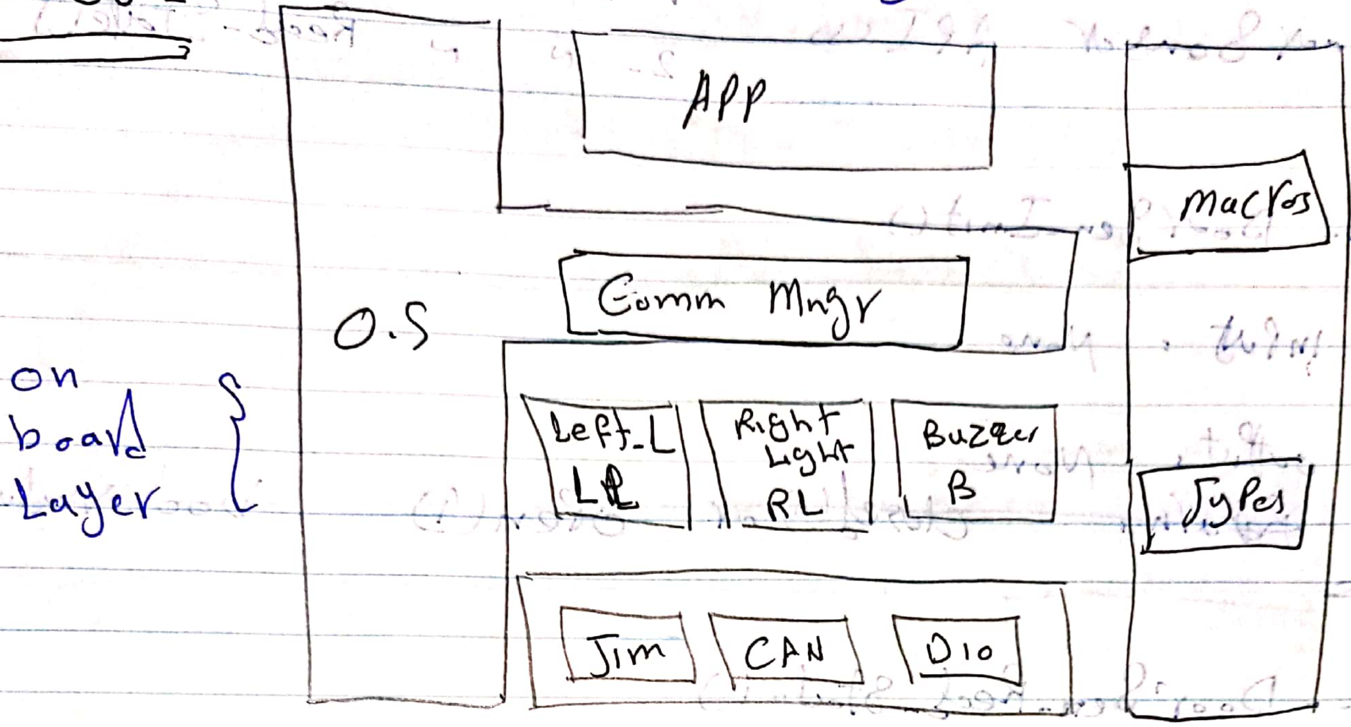
Return : closed (0) or open

input : None

output : None

For ~~Speed Sensor~~ speed Sensor and Light SW the API(s) will be the same.
So No need to write them also.

"ECU2" "Layers"



For most of the ~~APIs~~ modules, APIs are the same except (B, LL, RL)

For Buzzer

1. Buzzer-Init()
2. Turn On/off ()

Same thing for RL, LL

Buzzer_Init()

| | | |
|---|---|---|
| input | : | None |
| output | : | None |
| Return | : | True or False |
| function | : | Initializes Buzzer Module |

2- Turn on_Buz ()

3- Turn_off_Buz()

Input : None

Ack (1)
NAck (0)

Output : None      or

Return : uint8 "Buzzer State" or (void)

Func : Turn on /off Buzzer

You Can Implement 1 function taking ~~8~~ an integer value (uint8)

{ 0 → ——→ 0
Anything else → 1

For L_Light and R_Light functions are the same as Buzzer API(s)