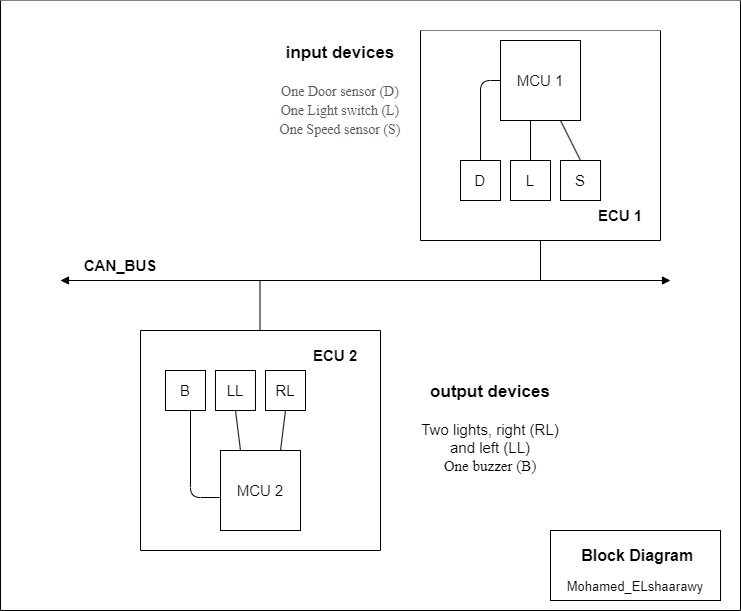
**Automotive door control system design**

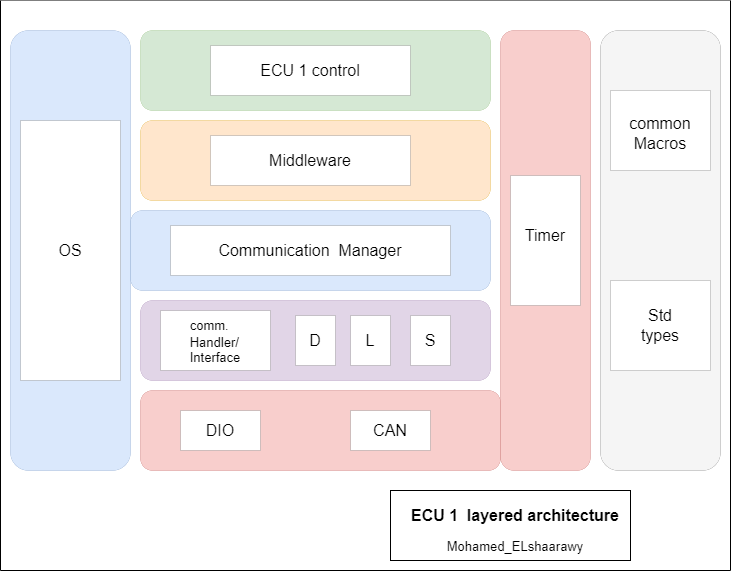
**Static design analysis**

* Block Diagram



For ECU 1:

**1.** Make the layered architecture



**2.** Specify ECU components and modules

* Application (ECU 1 control) : Where system logic and tasks are implemented.
* Main.c
* Middleware : For routing the system to its desired destination.
* Middleware.c , Middleware.h
* Services:
* OS : assume (FreeRTOS)

. all Kernel files

* Comm Manager(BCM) : used to select comm protocol and send status

. BCM.c , BCM.h

* Lib(Common): contain common header files
* Common Macros : comm\_Macros.h
* Std types : Std\_types.h
* HAL :
* Onboard layer :

. Door sensor (D) : D.c , D.h

. Light switch (L) : L.c , L.h

. Speed sensor (S) : S.c , S.h

* MCAL :

. DIO : DIO.c , DIO.h

. CAN : CAN.c , CAN.h

. Timer : Timer.c , Timer.h

**3.** Provide full detailed APIs for each module as well as a detailed description for the used typedefs

* Application module:
* Void App\_Init( );

. initialize ECU , call all Peripheral modules (init functions )

. Non-Reentrant, Synchronous, Non-Recursive Function.

* Void App\_start ( );

. All program logic goes here.

. Reentrant, Asynchronous, Non-Recursive Function

* void Callback(void);

.used with ISR to do some logic

. Non-Reentrant, Synchronous, Non-Recursive Function

* Middleware module:
* Void Middleware\_sendData( Protocole\_t protocole , data\_t data ) ;

. select comm manager ,comm protocole and send data ( status )

. Reentrant, Asynchronous, Non-Recursive Function.

* BCM module:
* BCM\_CANsend( data\_t data );

. use CAN protocole to send data ( status ).

. Reentrant, Asynchronous, Non-Recursive Function.

* Door Sensor( D ) module:
* D\_Init( D\_config\_t\* config );

. take sensor configurations and initialize door sensor

. Reentrant, Asynchronous, Non-Recursive Function.

* D\_state\_t D\_getState( );

. Return door status ( open || close )

. Non-Reentrant, Synchronous, Non-Recursive Function

* Light switch ( L ) module:
* L\_Init( L\_config\_t\* config );

. take switch configurations and initialize Light switch

. Reentrant, Asynchronous, Non-Recursive Function.

* L\_state\_t L\_getState( );

. Return switch status ( on || off )

. Non-Reentrant, Synchronous, Non-Recursive Function

* Speed sensor ( S ) module:
* S\_Init( S\_config\_t\* config );

. take sensor configurations and initialize Speed sensor

. Reentrant, Asynchronous, Non-Recursive Function.

* S\_value\_t S\_getValue( );

. Return speed value

. Non-Reentrant, Synchronous, Non-Recursive Function

* DIO module:
* DIO\_Init( DIO\_config\_t\* config);

. take DIO configurations and initialize DIO PORTs

. Reentrant, Asynchronous, Non-Recursive Function.

* DIO\_level\_t DIO\_read(DIO\_port\_t port , DIO\_pin\_t pin);

. Return pin value ( LOW=0 || HIGH=1 )

. Non-Reentrant, Asynchronous, Non-Recursive Function.

* DIO\_write(DIO\_port\_t port , DIO\_pin\_t pin, Uint8\_t value);

. Set pin value (( LOW=0 || HIGH=1 ))

. Non-Reentrant, Asynchronous, Non-Recursive Function.

* CAN module:
* Void CAN\_Init( CAN\_config\_t\* config );

. use CAN configurations to initialize CAN peripheral

. Reentrant, Asynchronous, Non-Recursive Function

* Void CAN\_send( data\_t data);

. Tack data to be send

. Non-Reentrant, Asynchronous, Non-Recursive Function

* Data\_t CAN\_receive( data\_t\* data);

. receive data

. Reentrant, Synchronous, Non-Recursive Function

* Timer module:
* Void Timer\_Init( Timer\_config\_t config);

. tack timer configurations to initialize timer peripheral

. Reentrant, Asynchronous, Non-Recursive Function

* Void Timer\_start();

. start timer after initialization

. Non-Reentrant, Asynchronous, Non-Recursive Function.

* Void timer\_stop();

. stop timer

. Non-Reentrant, Asynchronous, Non-Recursive Function.

* Void Timer\_delay\_ms(delay\_t delat);

. Delay time in ms

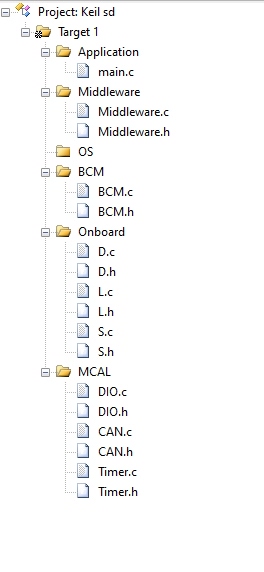
. Non-Reentrant, Asynchronous, Non-Recursive Function.

* Common Macros module:
* SET\_BIT(Port, Pin)
* CLEAR\_BIT(Port, Pin)
* READ\_BIT(PORT,PIN)

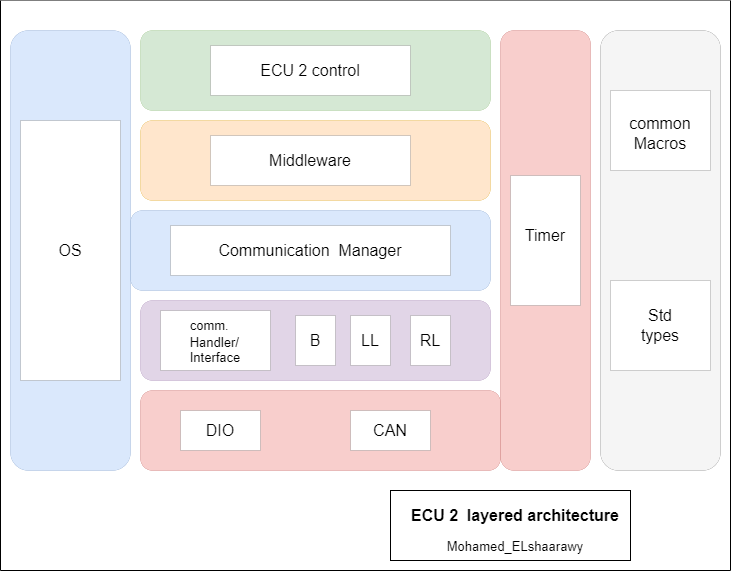
. Function-like macros to(set, clear , read )PINs

* used typedefs
* Protocole\_t : enum for protocol type (only CAN for this project )
* data\_t  : pre-defined type ( uint8\_t || uint16\_t …………..ext )
* D\_config\_t : struct for door sensor configuration
* D\_state\_t : enum for door status (open , close )
* L\_config\_t : struct for light switch configuration
* L\_state\_t : enum for light status (on , off )
* S\_config\_t : struct for speed sensor configuration
* S\_value\_t : pre-defined type ( uint8\_t || uint16\_t …………..ext )
* DIO\_config\_t : struct for DIO configuration
* DIO\_level\_t : enum for PIN level (LOW , HIGH)
* DIO\_port\_t : enum for PORT type (PORT\_!,PORT\_2,………ext)
* DIO\_pin\_t : enum for PIN type (PIN\_1 ,PIN\_2………ext )
* CAN\_config\_t : struct for CAN configuration
* Timer\_config\_t: struct for Timer configuration
* delay\_t : pre-defined type ( uint8\_t || uint16\_t …………..ext )

**4.** Prepare your folder structure according to the previous points



For ECU 2:

**1.** Make the layered architecture

**2.** Specify ECU components and modules

* Application (ECU 2 control) : Where system logic and tasks are implemented.
* Main.c
* Middleware : For routing the system to its desired destination.
* Middleware.c , Middleware.h
* Services:
* OS : assume (FreeRTOS)

. all Kernel files

* Comm Manager(BCM) : used to select comm protocol and send status

. BCM.c , BCM.h

* Lib(Common): contain common header files
* Common Macros : comm\_Macros.h
* Std types : Std\_types.h
* HAL :
* Onboard layer :

. Right Light (RL) : RL.c , LL.h

. Left Light (LL) : LL.c , LL.h

. Buzzer (B) : B.c , B.h

* MCAL :

. DIO : DIO.c , DIO.h

. CAN : CAN.c , CAN.h

. Timer : Timer.c , Timer.h

**3.** Provide full detailed APIs for each module as well as a detailed description for the used typedefs

* Application module:
* Middleware module:
* BCM module:
* Right Light ( RL ) module:
* Void RL\_Init( RL\_config\_t\* config );

. take RL configurations and initialize RL

. Reentrant, Asynchronous, Non-Recursive Function.

* Void RL\_on( );

. turn RL ON

. Non- Reentrant, Asynchronous, Non-Recursive Function.

* Void RL\_off( );

. turn RL OFF

. Non-Reentrant, Asynchronous, Non-Recursive Function.

* RL\_state\_t RL\_getState( );

. Return RL status ( ON || OFF )

. Non-Reentrant, Synchronous, Non-Recursive Function

* Left light ( LL ) module:
* Void RL\_Init( LL\_config\_t\* config );

. take LL configurations and initialize LL

. Reentrant, Asynchronous, Non-Recursive Function.

* Void LL\_on( );

. turn LL ON

. Non- Reentrant, Asynchronous, Non-Recursive Function.

* Void LL\_off( );

. turn LL OFF

. Non-Reentrant, Asynchronous, Non-Recursive Function.

* LL\_state\_t LL\_getState( );

. Return LL status ( ON || OFF )

. Non-Reentrant, Synchronous, Non-Recursive Function

* Buzzer ( B ) module:
* B\_Init( B\_config\_t\* config );

. take Buzzer configurations and initialize Buzzer

. Reentrant, Asynchronous, Non-Recursive Function.

* Void B\_on( );

. turn Buzzer ON

. Non- Reentrant, Asynchronous, Non-Recursive Function.

* Void B\_off( );

. turn Buzzer OFF

. Non-Reentrant, Asynchronous, Non-Recursive Function.

* DIO module:
* CAN module:
* Timer module:
* Common Macros module:
* Std types module

Unmentioned APIs are the same as ECU 1

* used typedefs
* RL\_config\_t : struct for Right Light configuration
* RL\_state\_t : enum for RL status (ON , OFF )
* LL\_config\_t : struct for Left Light configuration
* Ll\_state\_t : enum for LL status (ON , OFF )
* B\_config\_t : struct for Buzzer configuration

**4.** Prepare your folder structure according to the previous points

