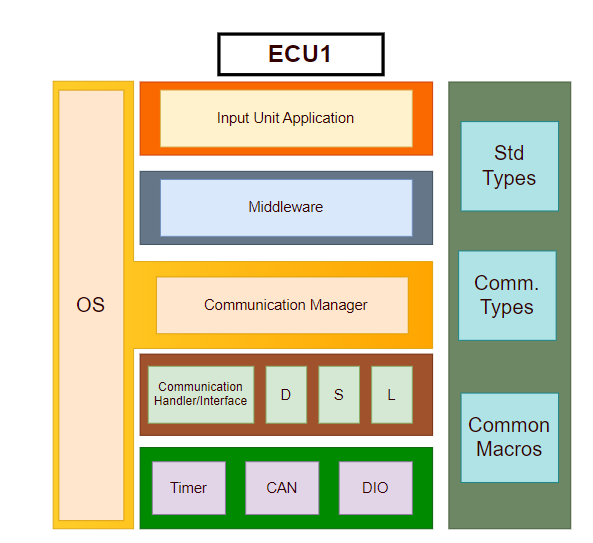
EgFWD

Advanced Embedded Systems

Project (3) | System Design

Automotive Door Control System Design Report

**Static Design**

**For ECU 1:**

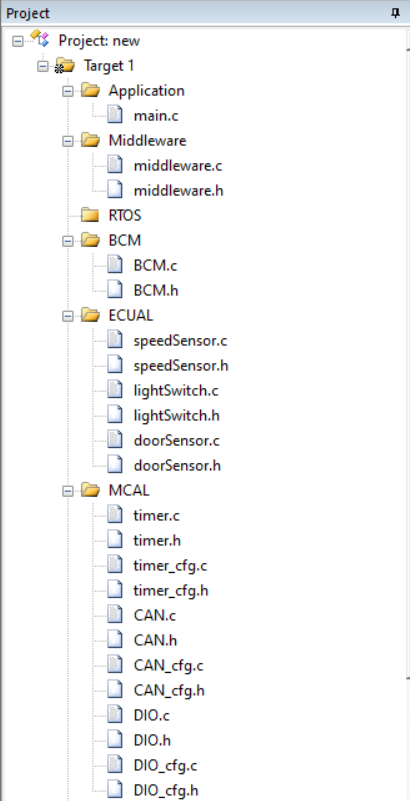
1. Make the layered architecture
2. Specify ECU components and modules
   1. Application: Where system logic and tasks are implemented.
      1. main.c
   2. Middleware: For routing the system to its desired destination. For example, either to send data from input devices and so being routed to communication manager or else saving this data in EEPROM and so being routed to memory manager.
      1. middleware.c | middleware.h
   3. Service:
      1. RTOS: Handling tasks and operating system as stated in project rubric.
         1. All RTOS related files from previous course(RTOS Porting Section | memory management, heap, tasks.c, config.h …etc)
      2. Communication(BCM): Used to send Status messages.
         1. BCM.c, BCM.h
   4. Common: For standard data types and common macros.
      1. std\_types.h
      2. common\_macros.h
      3. communication\_types.h
   5. HAL:
      1. Onboard/ECUAL: containing Door sensor, Light switch, Speed sensor and all external input devices.
         1. doorSensor.c, doorSensor.h
         2. lightSwitch.c, lightSwitch.h
         3. speedSensor.c, speedSensor.h
      2. MCAL:
         1. Timer
            1. timer.c, timer.h
            2. timer\_cfg.c, timer\_cfg.h
         2. CAN
            1. CAN.c, CAN.h
            2. CAN\_cfg.c, CAN\_cfg.h
         3. DIO
            1. DIO.c. DIO.h
            2. DIO\_cfg.c, DIO\_cfg.h
3. Provide full detailed APIs for each module as well as a detailed description for the used typedefs

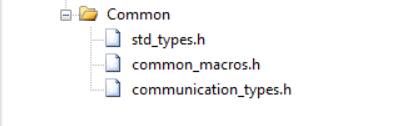
* Application Module:
  + API-Functions:
    - App\_InitError\_t App\_init ();
      * To create tasks and call all modules’ init APIs as GPIO, Timer…etc.
      * Non-Reentrant, Synchronous, Non-Recursive Function.
    - void App\_Start ();
      * All program logic goes here.
      * Reentrant, Asynchronous, Non-Recursive Function.
    - void timerSetCallback();
      * Non-Reentrant, Synchronous, Non-Recursive Function.
  + API-Types:
    - App\_InitError
      * To return initialization state, either success or fail.
* Middleware Module:
  + API-Functions:
    - Middleware\_SendState SendData (commProtocol\_t, uint32\_t data, dataSize\_t);
      * To send data using specified communication protocol.
      * Reentrant, Asynchronous, Non-Recursive Function.
    - Middleware\_SaveState SaveData (memoryType\_t, dataSize\_t, uint32\_t data);
      * To save data on internal EEPROM, external one or any other.
      * Reentrant, Asynchronous, Non-Recursive Function.
  + API-Types:
    - Middleware\_SaveState:
      * To return save state, either success or fail.
    - Middleware\_SendState:
      * To return send state, either success or fail.
* BCM Module:
  + API-Function:
    - Send\_CAN (CAN\_config\_ptr, dataSize\_t, uint32\_t data);
      * To route data to CAN module with specified configurations.
      * Reentrant, Asynchronous, Non-Recursive Function.
    - Send\_UART (UART\_config\_ptr, dataSize\_t, uint32\_t data);
      * To route data to UART module with specified configurations.
      * Reentrant, Asynchronous, Non-Recursive Function.
    - Send\_SPI (SPI\_config\_ptr, dataSize\_t, uint32\_t data);
      * To route data to SPI module with specified configurations.
      * Reentrant, Asynchronous, Non-Recursive Function.
    - Send\_I2C (I2C\_config\_ptr, dataSize\_t, uint32\_t data);
      * To route data to I2C module with specified configurations.
      * Reentrant, Asynchronous, Non-Recursive Function.
  + API-Types:
    - BCM\_SendState:
      * To return send state, either success or fail.
* Std\_Types Module:
  + API-Types: (mentioned below are just a few examples, feel free to add what’s needed)
    - uint8, uint16, uint32, int8, int16, int32…etc.
    - Boolean\_t
    - NULL
* Communication\_Types Module:
  + API-Types:
    - commProtocol\_t
      * struct for all communication protocols as SPI, UART…etc.
    - baudRate\_t
      * struct for all available baud rates.
    - parityType\_t
      * struct for either even or edd parity.
    - memoryType\_t
      * struct for internal, external EEPROM or any other memory.
    - dataSize\_t
      * size of data to be sent, could be a uint32\_t
* Common\_Macros Module:
  + API-Functions: (mentioned below are just a few examples, feel free to add what’s needed)
    - SetBit(Port, Pin)
      * Function-like macro.
    - ClearBit(Port, Pin)
      * Function-like macro.
    - ToggleBit(Port, Pin)
      * Function-like macro.
    - RotateRight(Port, n)
      * Function-like macro.
    - RotateLeft(Port, n)
      * Function-like macro.
* Timer Module:
  + API-Function:
    - Timer\_initState Timer\_init(Timer\_config\_ptr\_t);
      * Reentrant, Asynchronous, Non-Recursive Function.
    - void Timer\_Start();
      * Non-Reentrant, Synchronous, Non-Recursive Function.
    - void Timer\_Stop();
      * Non-Reentrant, Synchronous, Non-Recursive Function.
    - void setCallback();
      * To set ISR function passed from app layer.
      * Non-Reentrant, Synchronous, Non-Recursive ISR function.
    - uint32\_t Timer\_getElapsedTime();
      * returns time from beginning of timer’s counting.
      * Non-Reentrant, Synchronous, Non-Recursive Function.
    - uint32\_t Timer\_ getRemainingTime ();
      * returns remaining time from specified initial time to be counted.
      * Non-Reentrant, Asynchronous, Non-Recursive Function.
    - void Timer\_delay (uint32\_t);
      * counts in milliseconds
      * Non-Reentrant, Synchronous, Non-Recursive Function.
    - void Timer\_delayMicroseconds(uint32\_t);
      * Non-Reentrant, Synchronous, Non-Recursive Function.
  + API-Types:
    - Timer\_Length\_t
      * Normal or wide, 16 or 32 bit depending on architecture and available timers.
    - Timer\_Mode\_t
      * Normal, One-shot…etc.
    - Timer\_config\_ptr\_t
      * To specify certain configurations for timer/s used.
* CAN Module:
  + API-Functions
    - CAN\_initState CAN\_init(CAN\_config\_ptr\_t);
      * To initialize CAN Protocol.
      * Reentrant, Asynchronous, Non-Recursive Function.
    - void CAN\_Send(uint32\_t data, dataSize\_t);
      * Non-Reentrant, Synchronous, Non-Recursive Function.
    - void CAN\_Receive();
      * Non-Reentrant, Synchronous, Non-Recursive Function.
  + API-Types:
    - CAN\_sendState
      * Either sending success or fail.
    - CAN\_receiveState
      * Either receiving success or fail.
    - CAN\_initState
      * Initialization success or fail.
    - CAN\_config\_ptr\_t
* DIO Module:
  + API-Function:
    - Dio\_initState DIO\_init (DIO\_config\_ptr\_t);
      * To initialize different DIOs as input/output, ADC/Normal…etc.
      * Reentrant, Asynchronous, Non-Recursive Function.
    - uint8\_t DIO\_write (Port\_t, Pin\_t, LevelType\_t);
      * Non-Reentrant, Asynchronous, Non-Recursive Function.
    - uint8\_t DIO\_toggle (Port\_t, Pin\_t);
      * Non-Reentrant, Asynchronous, Non-Recursive Function.
    - uint8\_t DIO\_read (Port\_t, Pin\_t);
      * Non-Reentrant, Asynchronous, Non-Recursive Function.
    - Uint32\_t DIO\_readPort(Port\_t, Pin\_t);
      * Non-Reentrant, Asynchronous, Non-Recursive Function.
    - Uint32\_t DIO\_writePort(Port\_t);
      * Non-Reentrant, Asynchronous, Non-Recursive Function.
  + API-Types:
    - Port\_t: struct for Ports depending on microcontroller used.
    - Pin\_t: struct for pins from 0 to 8/16/32 depending on architecture.
    - LevelType\_t: either high or low.
    - Direction\_t: input or output.
    - AttachType\_t: PULL\_UP, PULL\_DOWN, OPEN\_DRAIN
    - PinType\_t: ADC, Normal…etc
    - Current\_t: how much can a pin withstand current, as in Arm 2,4,8mA
    - Dio\_config\_ptr\_t
    - Dio\_initState
      * Either initialization success or fail.

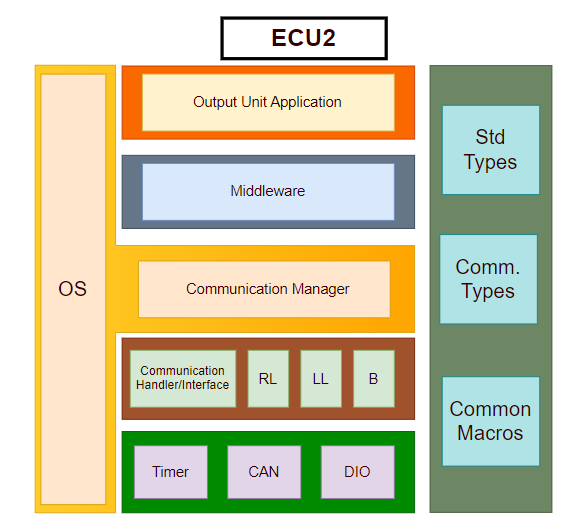
\*PREVIOUS MODULES ARE SAME FOR BOTH ECUs, NEXT 3 INPUT MODULES ARE WHAT’S DIFFERENT

* Door Sensor Module:
  + API-Function:
    - DoorSensor\_initState DoorSensor\_init(Port\_t, Pin\_t);
      * Initialize it as input pin.
      * Reentrant, Asynchronous, Non-Recursive Function.
    - LevelType\_t DoorState(Port\_t, Pin\_t);
      * Returns either door opened/closed with corresponding pin level.
      * Non-Reentrant, Synchronous, Non-Recursive Function.
  + API-Types:
    - Door\_initState
      * Either initialization success or fail.
* Light Switch Module:
  + API-Function:
    - LightSwitch\_initState LightSwitch\_init(Port\_T, Pin\_t);
      * Reentrant, Asynchronous, Non-Recursive Function.
    - LevelType\_t LightSwitchToggle (Port\_T, Pin\_t);
      * Non-Reentrant, Synchronous, Non-Recursive Function.
    - LevelType\_t LightSwitchState (Port\_t, Pin\_t);
      * Non-Reentrant, Synchronous, Non-Recursive Function.
  + API-Types:
    - LightSwitch\_initState
      * Either initialization success or fail.
* Speed Sensor Module:
  + API-Function:
    - SpeedSensor\_initState SpeedSensorInit(Port\_t, Pin\_t);
      * To initialize speed sensor pin as input.
      * Reentrant, Asynchronous, Non-Recursive Function.
    - uint32\_t SpeedSensorValue (void);
      * returns speed value read from speed sensor.
      * Non-Reentrant, Synchronous, Non-Recursive Function.
  + API-Types:
    - SpeedSensor\_initState
      * Either initialization success or fail.

1. Prepare your folder structure according to the previous points
   * + - Screenshot from Keil



****

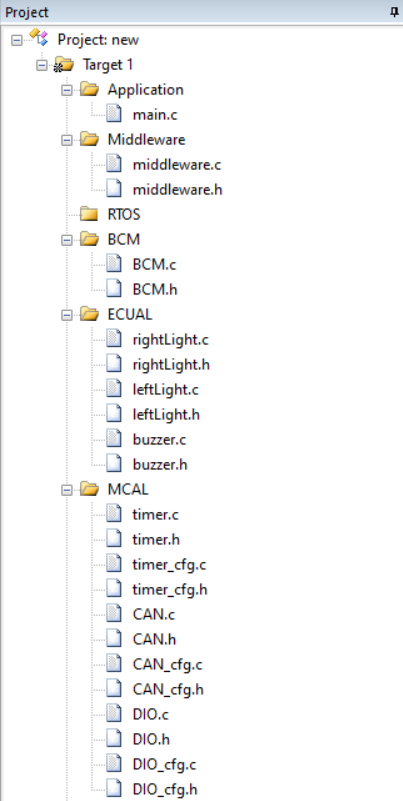
**For ECU 2:**

1. Make the layered architecture
2. Specify ECU components and modules
   1. Application: Where system logic and tasks are implemented.
   2. Middleware: For routing the system to its desired destination.
   3. RTOS: Handling tasks and operating system as stated in project rubric.
   4. Common: For standard data types and common macros.
   5. HAL:
      1. Onboard/ECUAL: containing Right Light , Left Light, Buzzer and all external output devices.
      2. MCAL: containing Timer, CAN and DIO modules.
3. Provide full detailed APIs for each module as well as a detailed description for the used typedefs

* Application Module
* Middleware Module
* BCM Module
* Std\_Types Module
* Common\_Macros Module
* Timer Module
* CAN Module
* DIO Module

\*PREVIOUS MODULES ARE SAME FOR BOTH ECUs, NEXT 3 OUTPUT MODULES ARE WHAT’S DIFFERENT

* Right Light Module:
  + API-Function:
    - RightLight \_initState RightLight \_init(Port\_t, Pin\_t);
      * Sets light as output.
      * Reentrant, Asynchronous, Non-Recursive Function.
    - Boolean\_t RightLightOn(Port\_t, Pin\_t);
      * Turns light on.
      * Non-Reentrant, Synchronous, Non-Recursive Function.
    - Boolean \_t RightLightOff(Port\_t, Pin\_t);
      * Turns light off.
      * Non-Reentrant, Synchronous, Non-Recursive Function.
    - LevelType\_t RightLightState(Port\_t, Pin\_t);
      * Returns light state either on or off according to corresponding level.
      * Non-Reentrant, Synchronous, Non-Recursive Function.
  + API-Types:
    - RightLight\_initState
      * Either initialization success or fail.
* Left Light Module:
  + API-Function:
    - LeftLight\_initState LeftLight\_init(Port\_t, Pin\_t);
      * Sets light as output.
      * Reentrant, Asynchronous, Non-Recursive Function.
    - Boolean\_t LeftLightOn (Port\_t, Pin\_t);
      * Turns light on.
      * Non-Reentrant, Synchronous, Non-Recursive Function.
    - Boolean \_t LeftLightOff(Port\_t, Pin\_t);
      * Turns light off.
      * Non-Reentrant, Synchronous, Non-Recursive Function.
    - LevelType\_t LeftLightState(Port\_t, Pin\_t);
      * Returns light state either on or off according to corresponding level.
      * Non-Reentrant, Synchronous, Non-Recursive Function.
  + API-Types:
    - LeftLight\_initState
      * Either initialization success or fail.
* Buzzer Module:
  + API-Function:
    - Buzzer\_initState Buzzer\_init(Port\_t, Pin\_t);
      * Sets buzzer pin as output.
      * Reentrant, Asynchronous, Non-Recursive Function.
    - Boolean\_t Buzzer\_On (void);
      * Turns buzzer on and returns Boolean value either action was taken or not.
      * Non-Reentrant, Synchronous, Non-Recursive Function.
    - Boolean\_t Buzzer\_Off (void);
      * Turns buzzer off and returns Boolean value either action was taken or not.
      * Non-Reentrant, Synchronous, Non-Recursive Function.
  + API-Types:
    - Buzzer\_initState
      * Either initialization success or fail.

1. Prepare your folder structure according to the previous points

