EgFWD

Advanced Embedded Systems

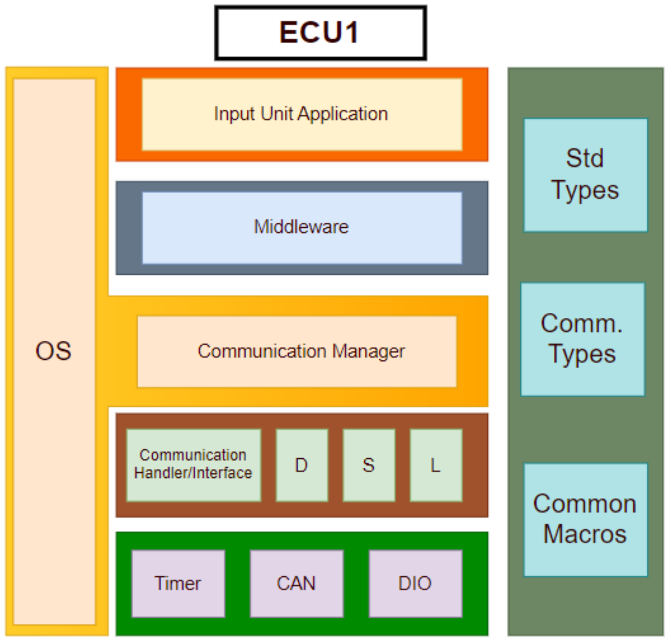
Project (3) | System Design

Automotive Door Control System Design Report

Static Design

For ECU 1:

1) Make the layered architecture



2) Specify ECU components and modules

a. Application: Where system logic and tasks are implemented.
    i. main.c
b. 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.
    i. middleware.c | middleware.h
c. Service:
    i. 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)
    ii. Communication(BCM): Used to send Status messages.
        1. BCM.c, BCM.h
d. Common: For standard data types and common macros.
    i. std_types.h
    ii. common_macros.h
    iii. communication_types.h
e. HAL:
    i. 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
    ii. MCAL:
        1. Timer

        a. timer.c, timer.h
        b. timer_cfg.c, timer_cfg.h
    2. CAN
        a. CAN.c, CAN.h
        b. CAN_cfg.c, CAN_cfg.h
    3. DIO
        a. DIO.c. DIO.h
        b. 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.
        - Takes no input arguments.
        - Returns initialization state, either success or fail.
        - Non-Reentrant, Synchronous, Non-Recursive Function.
    - ▪ void App_Start ();
        - All program logic goes here.
        - Takes no input arguments.
        - Returns nothing.
        - Reentrant, Asynchronous, Non-Recursive Function.
    - ▪ void timerSetCallback(void);
        - Function to pass timer ISR from application to timer layer.
        - Takes and returns nothing.
        - 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.
        - Takes desired communication protocol, actual data to be sent and its size as input arguments.

        - Returns sending state, either success or fail.
        - 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.
        - Takes type of memory for data to be saved in as internal or external EEPROM, actual data to be saved and its size as input arguments.
        - Returns save state, either success or fail.
        - 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-Functions:
    - Send_CAN (CAN_config_ptr, dataSize_t, uint32_t data);
        - To route data to CAN module with specified configurations.
        - Takes CAN configuration pointer to initialize CAN according to user preference, actual data to be sent and its size as input arguments.
        - Returns sending state, either success or fail.
        - Reentrant, Asynchronous, Non-Recursive Function.
    - Send_UART (UART_config_ptr, dataSize_t, uint32_t data);
        - To route data to UART module with specified configurations.
        - Takes UART configuration pointer to initialize UART according to user preference, actual data to be sent and its size as input arguments.
        - Returns sending state, either success or fail.
        - Reentrant, Asynchronous, Non-Recursive Function.
    - Send_SPI (SPI_config_ptr, dataSize_t, uint32_t data);
        - To route data to SPI module with specified configurations.
        - Takes SPI configuration pointer to initialize SPI according to user preference, actual data to be sent and its size as input arguments.
        - Returns sending state, either success or fail.
        - Reentrant, Asynchronous, Non-Recursive Function.
    - Send_I2C (I2C_config_ptr, dataSize_t, uint32_t data);

        - To route data to I2C module with specified configurations.
        - Takes I2C configuration pointer to initialize I2C according to user preference, actual data to be sent and its size as input arguments.
        - Returns sending state, either success or fail.
        - Reentrant, Asynchronous, Non-Recursive Function.
- API-Types:
    - BCM_SendState:
        - To be returned as 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
        - Struct holding TRUE and FALSE as they’re not one of the basic C types.
    - NULL
        - To be defined as zero.

❖ 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 odd 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 to set a bit to logic level 1.
    - ClearBit(Port, Pin)
        - Function-like macro to set a bit to logic level 0.
    - ToggleBit(Port, Pin)

        - Function-like macro to toggle a certain bit.
    - RotateRight(Port, n)
        - Function-like macro that takes Port address and number of times this Port would be right rotated.
    - RotateLeft(Port, n)
        - Function-like macro that takes Port address and number of times this Port would be left rotated.

❖ Timer Module:

- ➢ API-Functions:
    - Timer_initState Timer_init(Timer_config_ptr_t);
        - Initializes timer according to user preference passed through the configuration pointer.
        - Takes timer configuration pointer as an input argument.
        - Return timer initialization state, either success or fail.
        - Reentrant, Asynchronous, Non-Recursive Function.
    - void Timer_Start();
        - Starts timer that was initialized using previous init function.
        - Takes no input arguments.
        - Returns nothing.
        - Non-Reentrant, Synchronous, Non-Recursive Function.
    - void Timer_Stop();
        - Stops initiated timer.
        - Takes no input arguments.
        - Returns nothing.
        - Non-Reentrant, Synchronous, Non-Recursive Function.
    - void setCallback();
        - To set ISR function passed from app layer.
        - Takes and returns nothing; an ISR.
        - Non-Reentrant, Synchronous, Non-Recursive ISR function.
    - uint32_t Timer_getElapsedTime();
        - Takes no input arguments.
        - Returns time from beginning of timer’s counting.
        - Non-Reentrant, Synchronous, Non-Recursive Function.
    - uint32_t Timer_ getRemainingTime ();
        - Takes no input arguments.

        - Returns remaining time for timer to run out.
        - Non-Reentrant, Asynchronous, Non-Recursive Function.
    - void Timer_delay (uint32_t);
        - Counts in milliseconds
        - Takes delay time in milliseconds as an input argument.
        - Returns nothing.
        - Non-Reentrant, Synchronous, Non-Recursive Function.
    - void Timer_delayMicroseconds(uint32_t);
        - Counts in microseconds
        - Takes delay time in microseconds as an input argument.
        - Returns nothing.
        - 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);
        - Takes CAN configuration pointer to initialize CAN according to user preference.
        - Returns initialization state, either success or fail.
        - Reentrant, Asynchronous, Non-Recursive Function.
    - CAN_sendState CAN_Send(uint32_t data, dataSize_t);
        - Takes data to be sent and its size as input arguments.
        - Returns sending state, either success or fail.
        - Non-Reentrant, Synchronous, Non-Recursive Function.
    - CAN_receiveState CAN_Receive(uint32_t data);
        - Takes a uint32 to save received data in as a passed argument to it.
        - Returns receiving state, either success or fail.
        - 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-Functions:
    - Dio_initState DIO_init (DIO_config_ptr_t);
        - To initialize different DIOs as input/output, ADC/Normal…etc.
        - Takes user preference configuration pointer as input argument.
        - Return initialization state, either success or fail.
        - Reentrant, Asynchronous, Non-Recursive Function.
    - LevelType_t DIO_write (Port_t, Pin_t, LevelType_t);
        - To write a certain logic level on desired pin.
        - Takes Port and Pin to write on as input argument.
        - Returns level type written on pin, either high or low.
        - Non-Reentrant, Asynchronous, Non-Recursive Function.
    - LevelType_t DIO_toggle (Port_t, Pin_t);
        - Takes Port and Pin to toggle as input argument.
        - Returns level type written on pin, either high or low.
        - Non-Reentrant, Asynchronous, Non-Recursive Function.
    - LevelType_t DIO_read (Port_t, Pin_t);
        - Reads bit value on a certain pin.
        - Takes Port and Pin to be read as input argument.
        - Returns level type written on pin, either high or low.
        - Non-Reentrant, Asynchronous, Non-Recursive Function.
    - Uint32_t DIO_readPort(Port_t);
        - Reads value on a certain port.
        - Takes Port to be read as input argument.
        - Returns value read from port.
        - Non-Reentrant, Asynchronous, Non-Recursive Function.
    - Uint32_t DIO_writePort(Port_t);

        - Writes a certain value on desired port.
        - Takes Port to be written on as input argument.
        - Returns value written on port.
        - 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-Functions:
    - DoorSensor_initState DoorSensor_init(Port_t, Pin_t);
        - Initializes it as input pin.
        - Takes Port and Pin connected to this sensor as input arguments.
        - Return initialization state, either success or fail.
        - Reentrant, Asynchronous, Non-Recursive Function.
    - LevelType_t DoorState(Port_t, Pin_t);
        - Takes port and pin attached to this sensor as input arguments.
        - Returns either door opened/closed according to pin level.
        - Non-Reentrant, Synchronous, Non-Recursive Function.
- API-Types:
    - Door_initState
        - Either initialization success or fail.

❖ Light Switch Module:

- API-Functions:
    - LightSwitch_initState LightSwitch_init(Port_T, Pin_t);
        - Initializes light switch as an input pin.

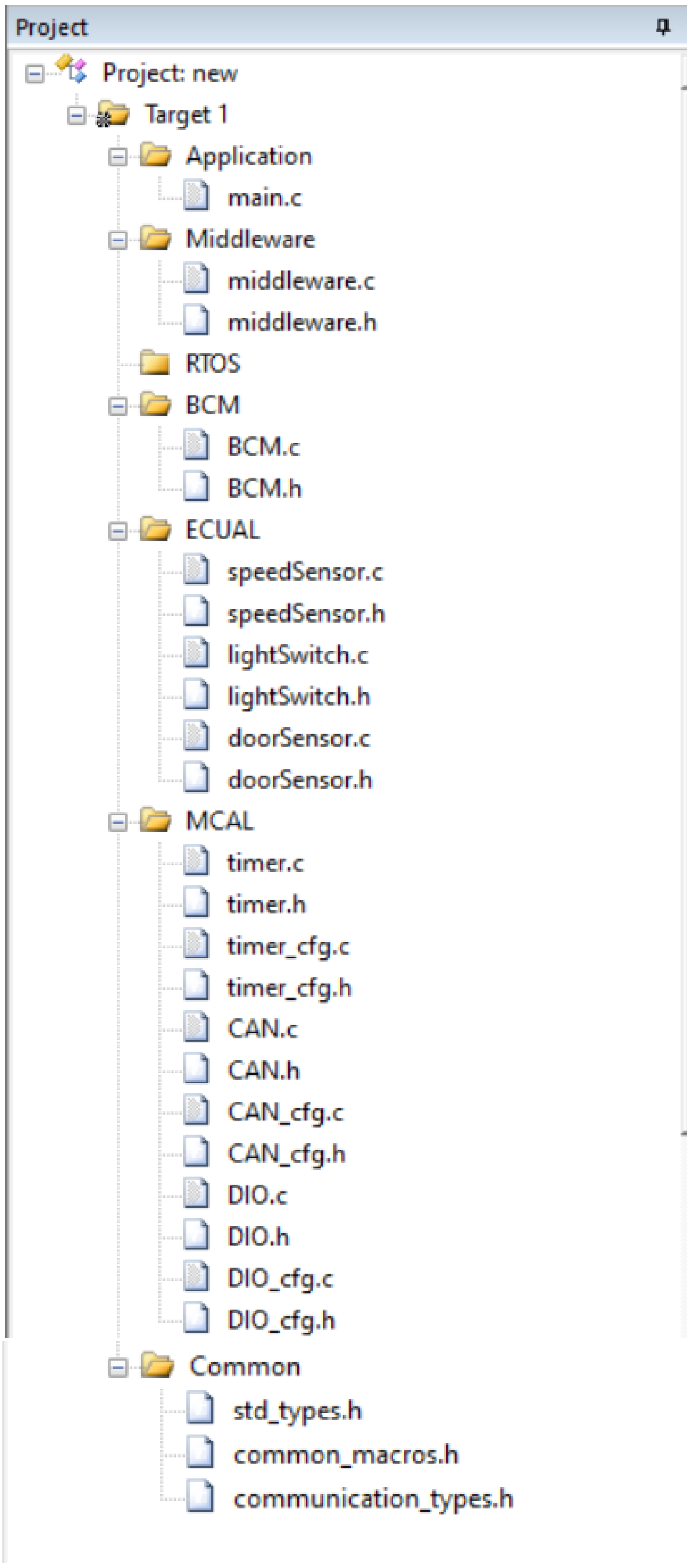
        - Takes port and pin attached to this switch as input arguments.
        - Returns initialization state, either success or fail.
        - Reentrant, Asynchronous, Non-Recursive Function.
    - LevelType_t LightSwitchState (Port_t, Pin_t);
        - Takes port and pin attached to this switch as input arguments.
        - Returns light switch state either pressed or not according to corresponding logic level on it.
        - Non-Reentrant, Synchronous, Non-Recursive Function.
- API-Types:
    - LightSwitch_initState
        - Either initialization success or fail.

❖ Speed Sensor Module:

- API-Functions:
    - SpeedSensor_initState SpeedSensorInit(Port_t, Pin_t);
        - To initialize speed sensor pin as input.
        - Takes port and pin attached to this sensor as input arguments.
        - Returns initialization state either success or fail.
        - Reentrant, Asynchronous, Non-Recursive Function.
    - uint32_t SpeedSensorValue (Port_t, Pin_t);
        - returns speed value read from speed sensor.
        - Takes port and pin attached to this switch as input arguments.
        - Non-Reentrant, Synchronous, Non-Recursive Function.
- API-Types:
    - SpeedSensor_initState
        - Either initialization success or fail.

4) Prepare your folder structure according to the previous points

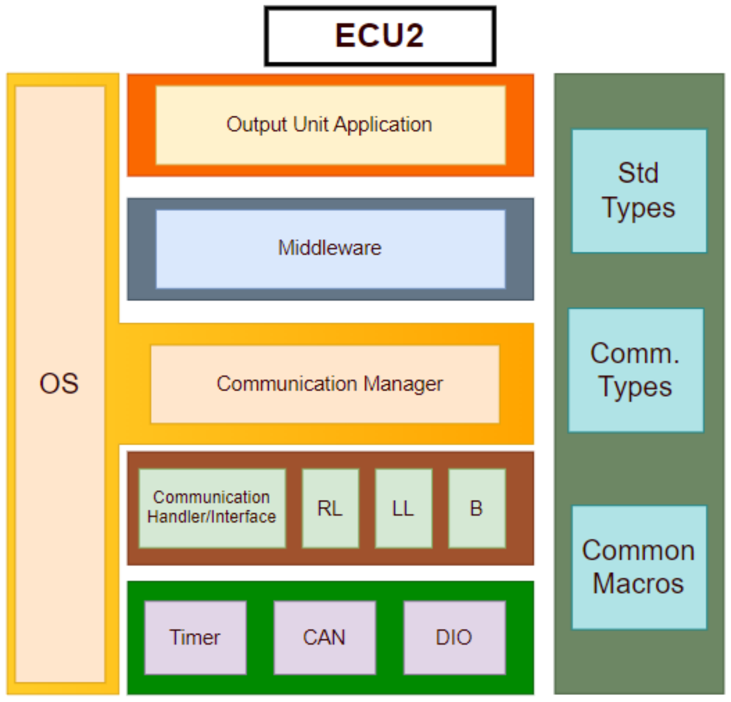
- Screenshot from Keil



```
Project
Project: new
  Target 1
    Application
      main.c
    Middleware
      middleware.c
      middleware.h
    RTOS
    BCM
      BCM.c
      BCM.h
    ECUAL
      speedSensor.c
      speedSensor.h
      lightSwitch.c
      lightSwitch.h
      doorSensor.c
      doorSensor.h
    MCAL
      timer.c
      timer.h
      timer_cfg.c
      timer_cfg.h
      CAN.c
      CAN.h
      CAN_cfg.c
      CAN_cfg.h
      DIO.c
      DIO.h
      DIO_cfg.c
      DIO_cfg.h
    Common
      std_types.h
      common_macros.h
      communication_types.h
```

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
    - ❖ Communication_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-Functions:
  - ▪ RightLight _initState RightLight _init(Port_t, Pin_t);
    - Sets light as output.
    - Takes port and pin attached to this light as input arguments.
    - Returns initialization state either success or fail.
    - Reentrant, Asynchronous, Non-Recursive Function.
  - ▪ Boolean_t RightLightOn(Port_t, Pin_t);
    - Turns light on.
    - Takes port and pin attached to this light as input arguments.
    - Returns state of this function, either done successfully(TRUE) or not(FALSE).
    - Non-Reentrant, Synchronous, Non-Recursive Function.
  - ▪ Boolean _t RightLightOff(Port_t, Pin_t);
    - Turns light off.
    - Takes port and pin attached to this light as input arguments.
    - Returns state of this function, either done successfully(TRUE) or not(FALSE).
    - Non-Reentrant, Synchronous, Non-Recursive Function.
  - ▪ LevelType_t RightLightState(Port_t, Pin_t);
    - Returns light state either on or off according to corresponding level on it.
    - Takes port and pin attached to this light as input arguments.
    - Non-Reentrant, Synchronous, Non-Recursive Function.
- ➢ API-Types:
  - ▪ RightLight_initState
    - Either initialization success or fail.

❖ Left Light Module:

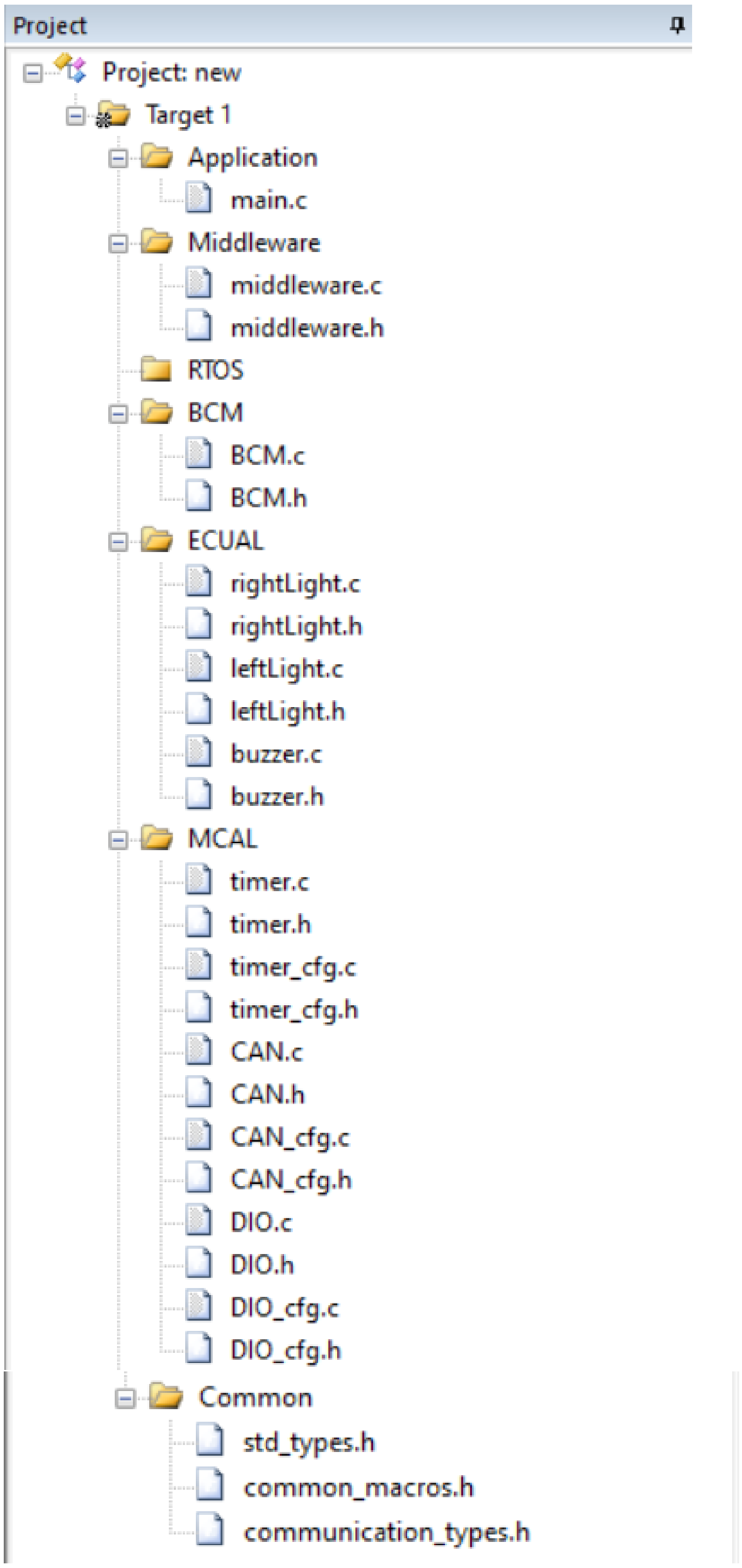
- ➢ API-Functions:
  - ▪ LeftLight_initState LeftLight_init(Port_t, Pin_t);
    - Sets light as output.
    - Takes port and pin attached to this light as input arguments.
    - Returns initialization state either success or fail.
    - Reentrant, Asynchronous, Non-Recursive Function.
  - ▪ Boolean_t LeftLightOn (Port_t, Pin_t);
    - Turns light on.
    - Takes port and pin attached to this light as input arguments.

        - Returns state of this function, either done successfully(TRUE) or not(FALSE).
        - Non-Reentrant, Synchronous, Non-Recursive Function.
    - Boolean _t LeftLightOff(Port_t, Pin_t);
        - Turns light off.
        - Takes port and pin attached to this light as input arguments.
        - Returns state of this function, either done successfully(TRUE) or not(FALSE).
        - Non-Reentrant, Synchronous, Non-Recursive Function.
    - LevelType_t LeftLightState(Port_t, Pin_t);
        - Returns light state either on or off according to corresponding level on it.
        - Takes port and pin attached to this light as input arguments.
        - Non-Reentrant, Synchronous, Non-Recursive Function.
- API-Types:
    - LeftLight_initState
        - Either initialization success or fail.

❖ Buzzer Module:

- API-Functions:
    - Buzzer_initState Buzzer_init(Port_t, Pin_t);
        - Sets buzzer pin as output.
        - Returns initialization state either success or fail.
        - Takes port and pin attached to this light as input arguments.
        - Reentrant, Asynchronous, Non-Recursive Function.
    - Boolean_t Buzzer_On (Port_t, Pin_t);
        - Turns buzzer on.
        - Returns Boolean value either action was done successfully(TRUE) or not(FALSE).
        - Takes port and pin attached to this light as input arguments.
        - Non-Reentrant, Synchronous, Non-Recursive Function.
    - Boolean_t Buzzer_Off (Port_t, Pin_t);
        - Turns buzzer off.
        - Returns Boolean value either action was done successfully(TRUE) or not(FALSE).
        - Takes port and pin attached to this light as input arguments.
        - Non-Reentrant, Synchronous, Non-Recursive Function.
- API-Types:
    - Buzzer_initState
        - Either initialization success or fail.

4. Prepare your folder structure according to the previous points



```
Project
Project: new
  Target 1
    Application
      main.c
    Middleware
      middleware.c
      middleware.h
    RTOS
    BCM
      BCM.c
      BCM.h
    ECUAL
      rightLight.c
      rightLight.h
      leftLight.c
      leftLight.h
      buzzer.c
      buzzer.h
    MCAL
      timer.c
      timer.h
      timer_cfg.c
      timer_cfg.h
      CAN.c
      CAN.h
      CAN_cfg.c
      CAN_cfg.h
      DIO.c
      DIO.h
      DIO_cfg.c
      DIO_cfg.h
    Common
      std_types.h
      common_macros.h
      communication_types.h
```