Automotive door and lights system design

Egypt Future work is digital

Contents

I. Overview of system components

A. ECU 1

ECU 1 is a microcontroller responsible for receiving data from three sensors

- The vehicle speed (Analog input with the speed value)
- The light switch (A digital input switch has two states on or off)
- Door state sensor (detect if the door is closed or open)

And resend the data received from the sensors to ECU 2 Via can bus

B. ECU 2

ECU 2 is a microcontroller responsible for controlling three devices

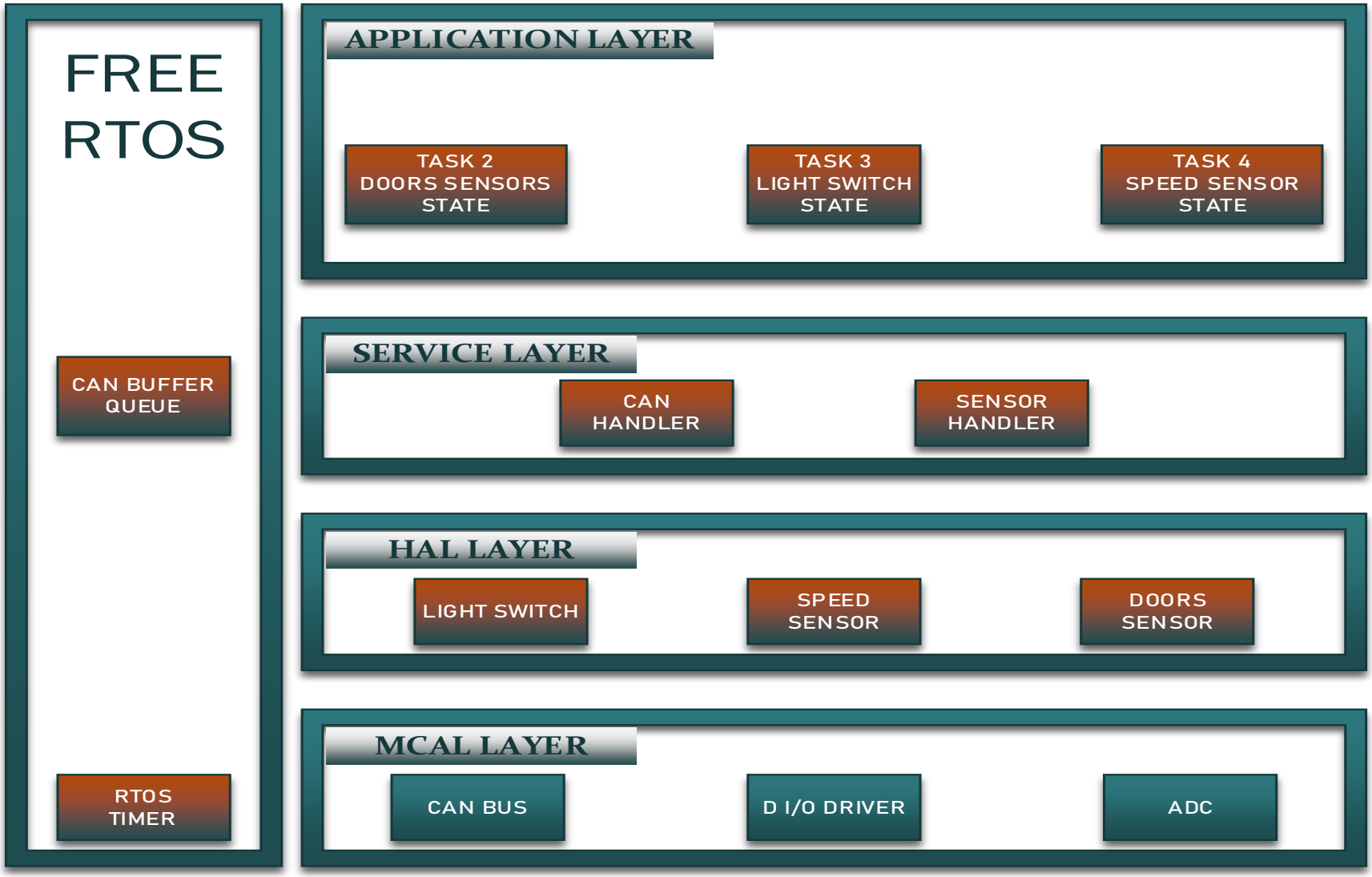
- Left lights (by a switch with two states ON/OFF)
- Right lights (by a switch with two states ON/OFF)
- The buzzer

ECU 2 also take the decision according to the received data from ECU.1

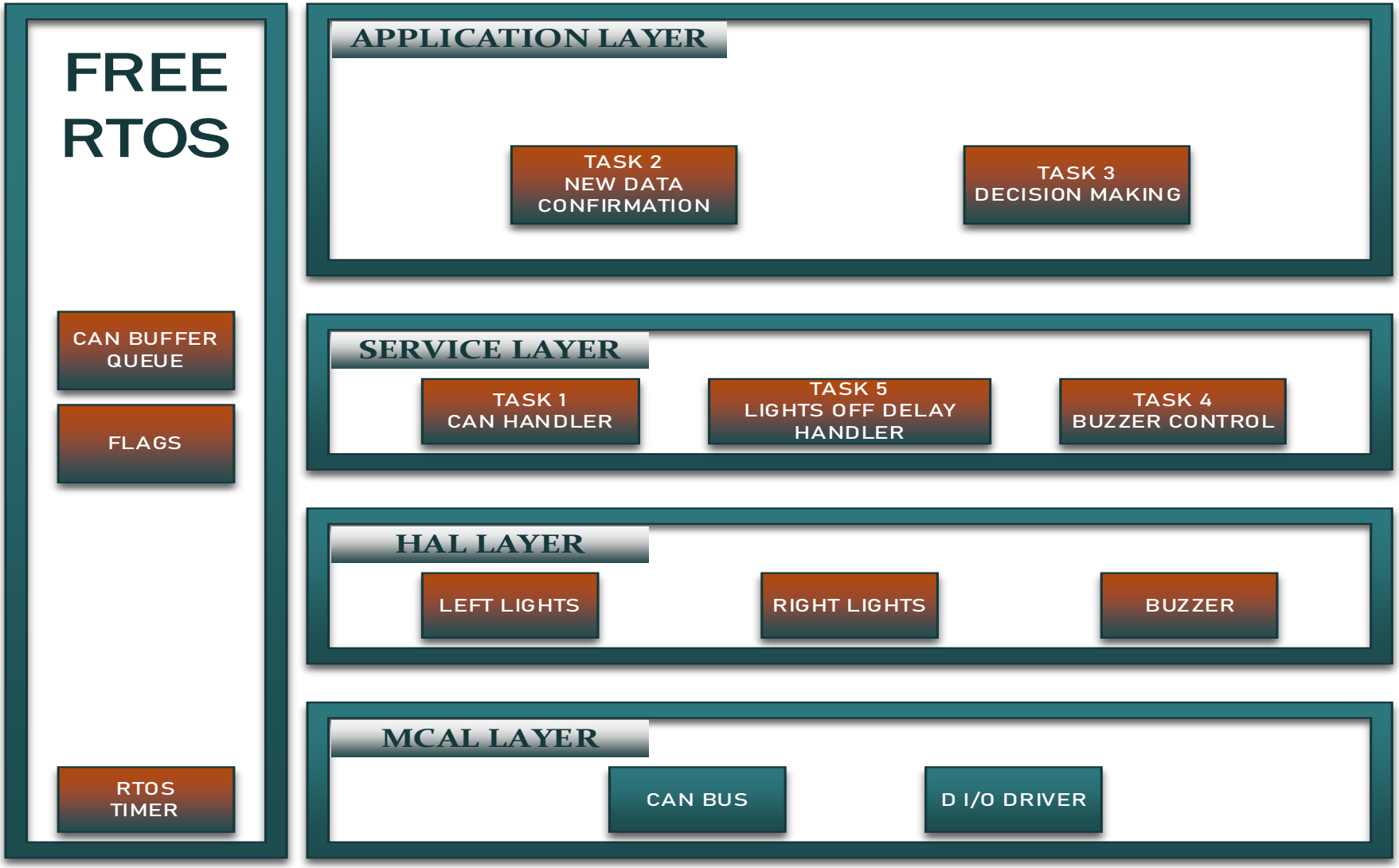
II. Layered architecture

The layered architecture for each ECU according to AUTOSAR principles

A. ECU 1 Layered architecture



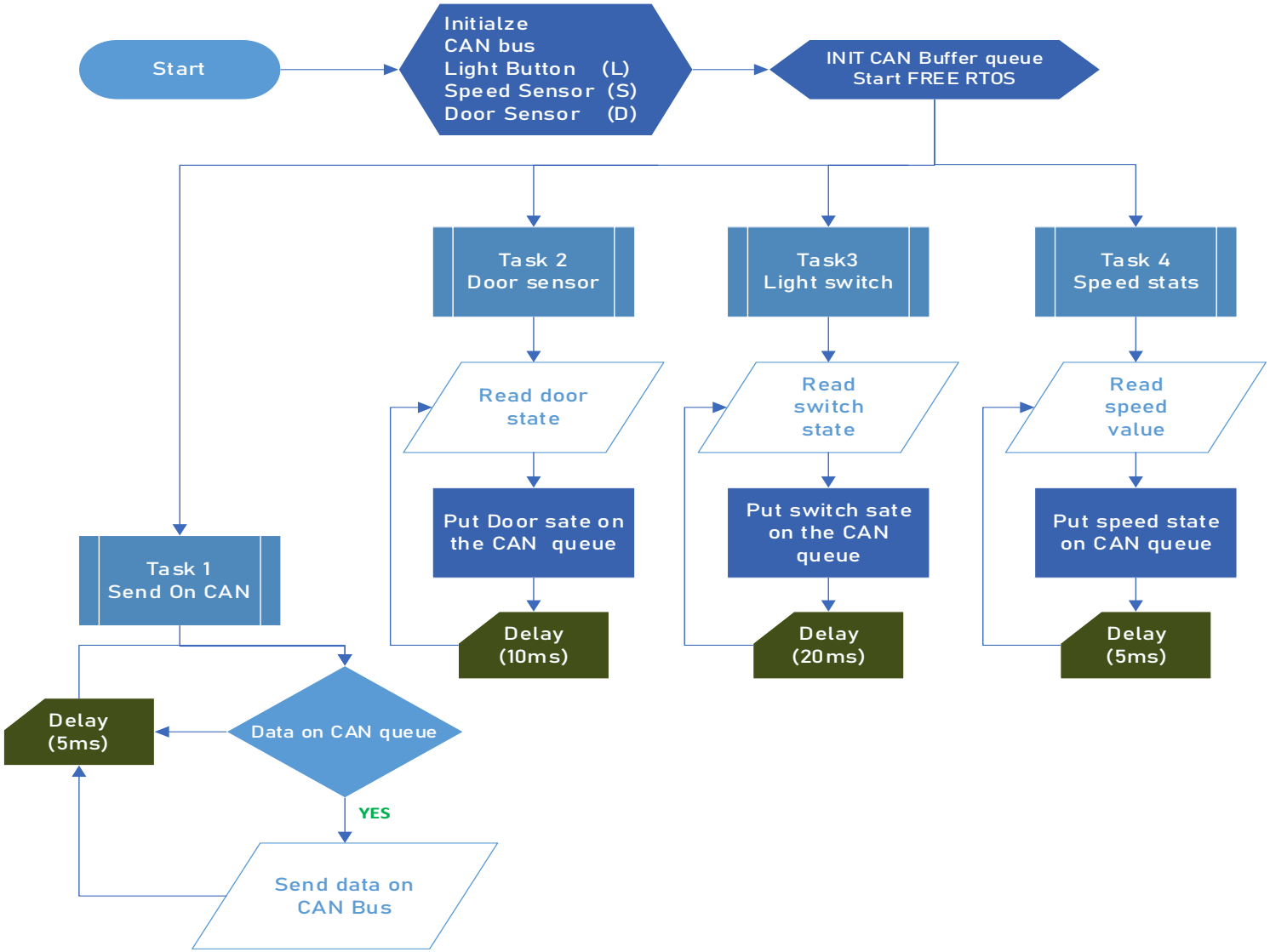
B. ECU 2 Layered architecture



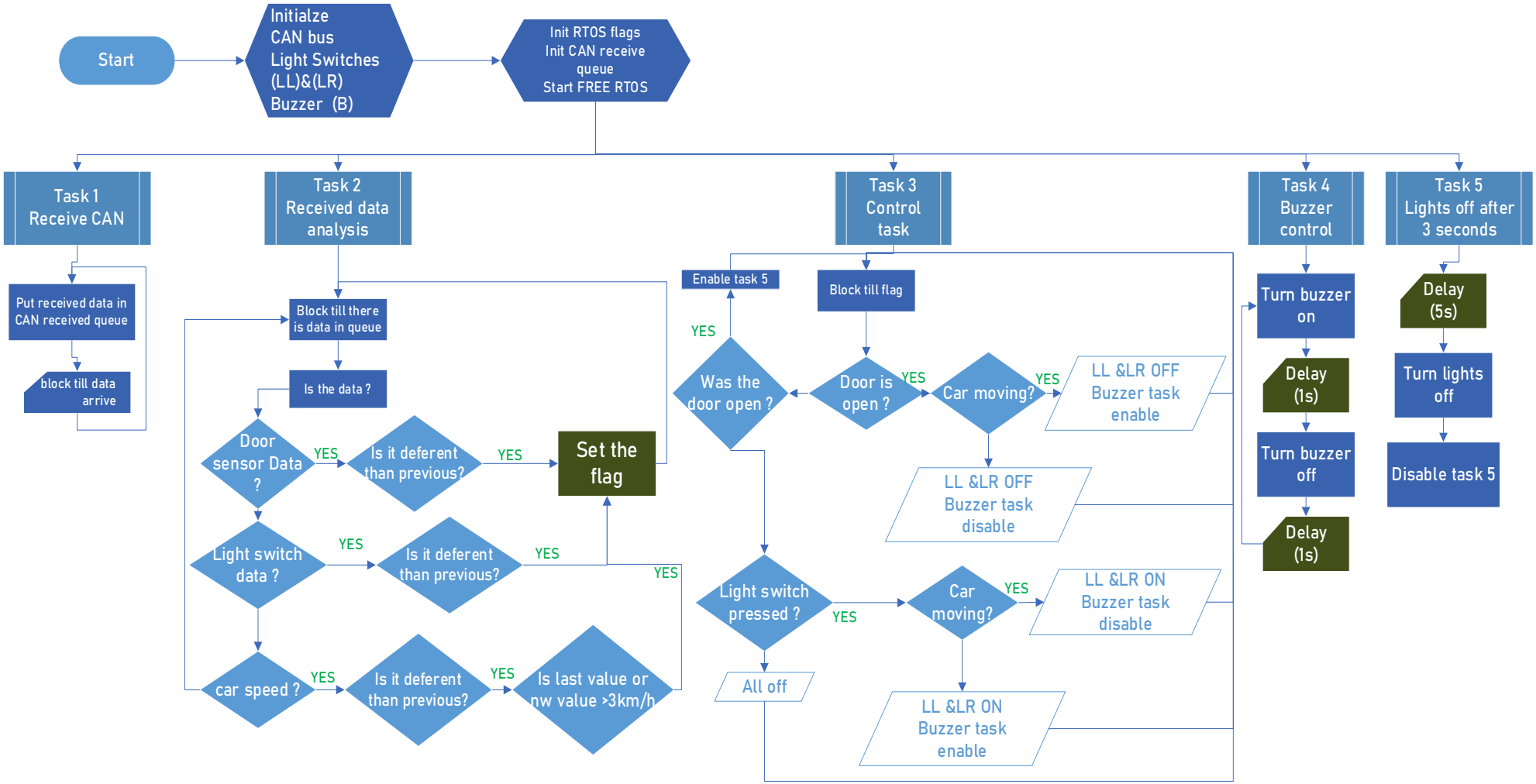
III. Flowcharts

Each flowchart explain the how the program will flow in each microcontroller used

A. ECU1 FlowChart



B. ECU 2 FlowChart



IV. Detailed API.s

Providing full-detailed API.s For each module in each layer for the microcontrollers used

A. ECU 1 API.s

For each layer Starting from down up → MCAL →HAL→Service → Application

1. MCAL (microcontroller abstraction layer)

1) MCAL CAN BUS

Function Name	CAN.Init()	
arguments	NON	
return	E_NOK	1
	E_OK	0
description	Initialize the CAN configuration from the CAN_CFG.h File	

Function Name	CAN.DeInit()	
arguments	NON	
return	E_NOK	1
	E_OK	0
description	Disable CAN Control over pins	

Function Name	CAN CAN_SetBaudrate(uint8 Controller, uint16 BaudRateConfig);	
arguments	uint8	uint16
	ControllerID	BaudRateConfig
return	E_NOK	1
	E_OK	0
description	Initialize the CAN configuration from the CAN_CFG.h File	

Function Name	CAN_Write(uint8 controller, char * Write_data_array);	
arguments	Char *	Uiunt8
	Data_array	controllerID
return	E_NOK	1
	E_OK	0
description	Write the data in the array on the bus	

Function Name	CAN_Read(uint8 controller, char * Read_data_array);	
arguments	Char *	Uiunt8
	Read_Data_array	controllerID
return	E_NOK	1
	E_OK	0
description	Put the data available in the array given	

Function Name	CAN_EnableControllerInterrupts(uint8 Controller);	
arguments	uint8	
	controller	
return	E_NOK	1
	E_OK	0
description	Enable the can interrupts	

Function Name	CAN_DisableControllerInterrupts(uint8 Controller);	
arguments	uint8	
	controller	
return	E_NOK	1
	E_OK	0

description	Disable the can interrupts

2) DI/O DRIVER

Function Name	DIO_init();	
arguments	NON	
	NON	
return	E_NOK	1
	E_OK	0
description	Initialize the GPIOs DIO as set in the configuration files	

Function Name	DIO_Deinit();	
arguments	NON	
	NON	
return	E_NOK	1
	E_OK	0
description	Disable the GPIOs peripheral control over the general purpose pins	

Function Name	DIO_Read(uint8 port,uint8 pin);	
arguments	Uint8	Pin number
	Uint8	Port number
return	Uint8	Pin value
description	Read the Pin value	

Function Name	DIO_write(uint8 port,uint8 pin,uint8 value);	
arguments	Uint8	pin
	Uint8	Port number
return	E_NOK	1
	E_OK	0
description	Write the given value on the pin	

Function Name	DIO_Write.Port(uint8 port,uint8 value);	
arguments	Uint8	pin
	Uint8	value
return	E_NOK	1
	E_OK	0
description	Write the given value on the port	

Function Name	DIO_Read(uint8 port,uint8 pin);	
arguments	Uint8	Port number
return	Uint8	Port value
description	Read the Pin value	

3) ADC DRIVER

Function Name	ADC_init();	
arguments	NON	
	NON	
return	E_NOK	1
	E_OK	0
description	Initialize the ADCs pins as set in the configuration files	

Function Name	ADC_DeInit();	
arguments	NON	
	NON	
return	E_NOK	1
	E_OK	0
description	Disable the ADC peripheral control over the GPIO's pins	

Function Name	ADC_read_10bit(uint8 port , uint8 pin);	
Arguments	uint8	Port
	uint8	Pin
Return	Uint16	ADC RESULT
Description	Start ADC conversion and return the result	

2. HAL (hardware abstraction layer)

1) Speed_sensor

Function Name	SpeedSensor.Init(uint8 port,uint8 pin);	
arguments	Uint8	pin
	Uint8	Port number
return	E_NOK	1
	E_OK	0
description	Initialize the speed sensor input as the given pin	

Function Name	Speed.Get();	
arguments	Uint8	pin
	Uint8	Port number
return	float	Return the speed value in km/hour
description	Get the ADC value of the sensor	

2) Door state sensor

Function Name	DoorSensorInit(uint8 port,uint8 pin);	
arguments	Uint8	pin
	Uint8	Port number
return	E_NOK	1
	E_OK	0
description	Initialize the Door sensor input as the given pin	

Function Name	Door.Get();	
arguments	Uint8	pin
	Uint8	Port number
return	bool	Value
description	Get the current door state	

3) Switch sense

Function Name	SwitchInit(uint8 port,uint8 pin);

arguments	Uint8	pin
	Uint8	Port number
return	E_NOK	1
	E_OK	0
description	Initialize the Switch input as the given pin	

Function Name	Sensor.SwitchGet();	
arguments	Uint8	pin
	Uint8	Port number
return	bool	Value
description	Get the current switch state	

3. Service layer

1) CAN Buffer queue

Create a queue to send the data to the CAN bus by it

2) CAN Handler

The CAN handler takes what it needs from the CAN driver , run in the background as a task in rtos to get the data sent for the controller by CAN bus

Function Name	CAN.On(uint8 controller);	
Arguments	uint8	
	controller	
Return	E_NOK	1
	E_OK	0

Description	Initialize the can controller as in configuration file and start its interrupt

Function Name	CAN.Off(uint8 controller);	
Arguments	uint8	
	controller	
Return	E_NOK	1
	E_OK	0
Description	Disable the can controller as in configuration file and its interrupt	

Function Name	CAN.Read(uint8 controller,uint16* ArrSize);	
Arguments	uint8	
	controller	
Return	Uint16 *	Uint16*
	Data	Arrsize
Description	Return a pointer to an array contain the data and put the size of the array in the arrsize value of the given address ,(return by address)	

Function Name	CAN.Write(uint8 controller,uint16* DataArr , uint16 ArrSize);		
Arguments	uint8	Uint16*	uint16
	controller	DataArr	ArrSize
Return	E_NOK	1	
	E_OK	0	
Description	Disable the can controller as in configuration file and its interrupt		

3) Sensor handler

Use the sensors APIs from the HAL layer to return the data required

Function Name	Sensor.ConfInit();	
arguments	NON	
return	E_NOK	1
	E_OK	0
description	Initialize sensor pins as in the configuration files	

Function Name	Sensor.SpeedInit(uint8 port,uint8 pin);	
arguments	Uint8	pin
	Uint8	Port number
return	E_NOK	1
	E_OK	0
description	Initialize the speed sensor input as the given pin and start its reading task	

Function Name	Sensor.DoorInit(uint8 port,uint8 pin);	
arguments	Uint8	pin
	Uint8	Port number
return	E_NOK	1
	E_OK	0
description	Initialize the Door sensor input as the given pin and start its corresponding task	

Function Name	Sensor.SwitchInit(uint8 port,uint8 pin);

arguments	Uint8	pin
	Uint8	Port number
return	E_NOK	1
	E_OK	0
description	Initialize the Switch input as the given pin and start its corresponding task	

Function Name	Sensor.SpeedGet();	
arguments	Uint8	pin
	Uint8	Port number
return	float	Return the speed value in km/hour
description	Initialize the Switch input as the given pin and start its corresponding task	

Function Name	Sensor.DoorGet();	
arguments	Uint8	pin
	Uint8	Port number
return	bool	Vlaue
description	Get the current door state	

Function Name	Sensor.SwitchGet();	
arguments	Uint8	pin
	Uint8	Port number
return	bool	Value
description	Get the current switch state	

4. Application layer

1) Door State task

Implement a task that sends the door state periodically every 10 milli-seconds via CANBus using the provided APIs

2) Speed sensor task

Implement a task that sends the door state periodically every 5 milli-seconds CANBus using the provided APIs

3) Switch state

Implement a task that sends the door state periodically every 20 milli-seconds CANBus using the provided APIs

B. MCU 2 APIs

For each layer Starting from down up → MCAL →HAL→Service → Application

1. MCAL (microcontroller abstraction layer)

1) MCAL CAN BUS

Function Name	CAN.Init()	
arguments	NON	
return	E_NOK	1
	E_OK	0
description	Initialize the CAN configuration from the CAN_CFG.h File	

Function Name	CAN.DeInit()	
arguments	NON	
return	E_NOK	1
	E_OK	0
description	Disable CAN Control over pins	

Function Name	CAN CAN_SetBaudrate(uint8 Controller, uint16 BaudRateConfig);

arguments	uint8	uint16
	ControllerID	BaudRateConfig
return	E_NOK	1
	E_OK	0
description	Initialize the CAN configuration from the CAN_CFG.h File	

Function Name	CAN_Write(uint8 controller, char * Write_data_array);	
arguments	Char *	Uiunt8
	Data_array	controllerID
return	E_NOK	1
	E_OK	0
description	Write the data in the array on the bus	

Function Name	CAN_Read(uint8 controller, char * Read_data_array);	
arguments	Char *	Uiunt8
	Read_Data_array	controllerID
return	E_NOK	1
	E_OK	0
description	Put the data available in the array given	

Function Name	CAN_EnableControllerInterrupts(uint8 Controller);	
arguments	uint8	
	controller	
return	E_NOK	1
	E_OK	0
description	Enable the can interrupts	

Function Name	CAN_DisableControllerInterrupts(uint8 Controller);	
arguments	uint8	
	controller	
return	E_NOK	1
	E_OK	0
description	Disable the can interrupts	

2) DI/O DRIVER

Function Name	DIO_init();	
arguments	NON	
	NON	
return	E_NOK	1
	E_OK	0
description	Initialize the GPIOs DIO as set in the configuration files	

Function Name	DIO_Deinit();	
arguments	NON	
	NON	
return	E_NOK	1
	E_OK	0
description	Disable the GPIOs peripheral control over the general purpose pins	

Function Name	DIO_Read(uint8 port,uint8 pin);

arguments	Uint8	Pin number
	Uint8	Port number
return	Uint8	Pin value
description	Read the Pin value	

Function Name	DIO_write(uint8 port,uint8 pin,uint8 value);	
arguments	Uint8	pin
	Uint8	Port number
return	E_NOK	1
	E_OK	0
description	Write the given value on the pin	

Function Name	DIO_Write.Port(uint8 port,uint8 value);	
arguments	Uint8	pin
	Uint8	value
return	E_NOK	1
	E_OK	0
description	Write the given value on the port	

Function Name	DIO_Read(uint8 port,uint8 pin);	
arguments	Uint8	Port number
return	Uint8	Port value
description	Read the Pin value	

2. HAL (hardware abstraction layer)

1) Left lights

Function Name	LeftLightsOn();	
arguments	NON	
	NON	
return	E_NOK	1
	E_OK	0
description	Turn on left lights	

Function Name	LeftLightsOff();	
arguments	NON	
	NON	
return	E_NOK	1
	E_OK	0
description	Turn off left lights	

Function Name	LeftLightsToggle();	
arguments	NON	
	NON	
return	E_NOK	1
	E_OK	0
description	Toggle left lights	

2) Right lights

Function Name	RightLightsOn();	
arguments	NON	

	NON	
return	E_NOK	1
	E_OK	0
description	Turn on Right lights	

Function Name	RightLightsOff();	
arguments	NON	
	NON	
return	E_NOK	1
	E_OK	0
description	Turn off Right lights	

Function Name	RightLightsToggle();	
arguments	NON	
	NON	
return	E_NOK	1
	E_OK	0
description	Toggle Right lights	

3) Buzzer

Function Name	BuzzerHigh();	
arguments	NON	
	NON	
return	E_NOK	1
	E_OK	0
description	Put high on the buzzer pin	

Function Name	BuzzerLow();	
arguments	NON	
	NON	
return	E_NOK	1
	E_OK	0
description	Put low the buzzer pin	

3. Service layer

1) CAN Handler

The CAN handler takes what it needs from the CAN driver , run in the background as a task in RTOS to get the data sent for the controller by CAN bus

Function Name	CAN.On(uint8 controller);	
arguments	uint8	
	controller	
return	E_NOK	1
	E_OK	0
description	Initialize the can controller as in configuration file and start its interrupt	

Function Name	CAN.Off(uint8 controller);	
arguments	uint8	
	controller	
return	E_NOK	1
	E_OK	0
description	Disable the can controller as in configuration file and its interrupt	

Function Name	CAN.Read(uint8 controller,uint16* ArrSize);	
arguments	uint8	
	controller	
return	Uint16 *	Uint16*
	Data	Arrsize
description	Return a pointer to an array contain the data and put the size of the array in the arrsize value of the given address ,(return by address)	

Function Name	CAN.Write(uint8 controller,uint16* DataArr , uint16 ArrSize);		
arguments	uint8	Uint16*	uint16
	controller	DataArr	ArrSize
return	E_NOK	1	
	E_OK	0	
description	Disable the can controller as in configuration file and its interrupt		

2) BUZZER control

Buzzer task is a task of a endless lop the make the repetitive sound of the buzzer

Function Name	Buzzer.Init()	
arguments	NON	
return	E_NOK	1
	E_OK	0
description	Initialize the buzzer	

Function Name	BuzzerOn();

arguments	NON	
	NON	
return	E_NOK	1
	E_OK	0
description	Enable buzzer task	

Function Name	BuzzerOff();	
arguments	NON	
	NON	
return	E_NOK	1
	E_OK	0
description	Suspend buzzer task	

3) Light off delay handler

Function Name	LightOffAfterDelay(uint64 delay);	
arguments	NON	
	NON	
return	E_NOK	1
	E_OK	0
description	Turn off the lights after Delay	

4. Application layer

1) New data confirmation

confirm that a new data has came to the microcontroller

2) make decision

Make the desired decision according the the new data that came from ECU2

V. Folders structures

C. ECU 1 Folder structure

1. MCAL
    - 1.1. Inc
        - 1.1.1. Can_cfg.h
        - 1.1.2. Can_std_types.h
        - 1.1.3. Can.h
        - 1.1.4. Dio_std_types.h
        - 1.1.5. Dio_cfg.h
        - 1.1.6. Dio.h
        - 1.1.7. ADC_cfg.h
        - 1.1.8. ADC_Stdtypes.h
        - 1.1.9. ADC.h
    - 1.2. Source
        - 1.2.1. Can.c
        - 1.2.2. Can_Lcfg.c
        - 1.2.3. Adc.c
        - 1.2.4. Adc_Lcfg.c
        - 1.2.5. Dio.c
        - 1.2.6. Dio_cfg.c
2. HAL
    - 2.1. Inc
        - 2.1.1. LightSwitch.h
        - 2.1.2. LightSwitch_cfg.h
        - 2.1.3. LightSwitch_std_types.h
        - 2.1.4. SpeedSensor_cfg.h
        - 2.1.5. SpeedSensor.h
        - 2.1.6. SpeedSensorstd_types.h
        - 2.1.7. DoorSensor.h
        - 2.1.8. DoorSensor_std_types.h
        - 2.1.9. DoorSensor_cfg
    - 2.2. Source
        - 2.2.1. DoorSensor.c
        - 2.2.2. DoorSensor _Lcfg.c
        - 2.2.3. SpeedSensor.c
        - 2.2.4. SpeedSensor _Lcfg.c
        - 2.2.5. LightSwitch.c
        - 2.2.6. LightSwitch _cfg.c
3. SERVICE
    - 3.1. Inc
        - 3.1.1. CanHandler_cfg.h
        - 3.1.2. CanHandler_std_types.h
        - 3.1.3. CanHandler.h
        - 3.1.4. SensorHandler_types.h
        - 3.1.5. SensorHandler _cfg.h
        - 3.1.6. SensorHandler.h
    - 3.2. Source
        - 3.2.1. CanHandler.c

3.2.2. CanHandler_Lcfg.c
3.2.3. SensorHandler.c
3.2.4. SensorHandler _cfg.c

A. ECU 2 Folder structure

1. MCAL
   1.1. Inc
      1.1.1. Can_cfg.h
      1.1.2. Can_std_types.h
      1.1.3. Can.h
      1.1.4. Dio_std_types.h
      1.1.5. Dio_cfg.h
      1.1.6. Dio.h
   1.2. Source
      1.2.1. Can.c
      1.2.2. Can_Lcfg.c
      1.2.3. Adc.c
      1.2.4. Adc_Lcfg.c
      1.2.5. Dio.c
      1.2.6. Dio_cfg.c
2. HAL
   2.1. Inc
      2.1.1. LeftLight.h
      2.1.2. LeftLight _cfg.h
      2.1.3. LeftLight_std_types.h
      2.1.4. RightLights_cfg.h
      2.1.5. RightLights.h
      2.1.6. RightLights_std_types.h
      2.1.7. Buzzer.h
      2.1.8. Buzzer_std_types.h
      2.1.9. Buzzer_cfg
   2.2. Source
      2.2.1. RightLights.c
      2.2.2. RightLights_Lcfg.c
      2.2.3. Buzzer.c
      2.2.4. Buzzer_Lcfg.c
      2.2.5. LeftLight.c
      2.2.6. LeftLight_cfg.c
3. SERVICE
   3.1. Inc
      3.1.1. CanHandler_cfg.h
      3.1.2. CanHandler_std_types.h
      3.1.3. CanHandler.h
      3.1.4. LightsoffDelay_types.h
      3.1.5. LightsoffDelay _cfg.h
      3.1.6. LightsoffDelay.h
      3.1.7. BuzzerControl_cfg.h
      3.1.8. BuzzerControl _Stdtypes.h
      3.1.9. BuzzerControl.h
   3.2. Source
      3.2.1. CanHandler.c

3.2.2. CanHandler_Lcfg.c
3.2.3. BuzzerControl.c
3.2.4. BuzzerControl _Lcfg.c
3.2.5. LightsoffDelay.c
3.2.6. LightsoffDelay _cfg.c