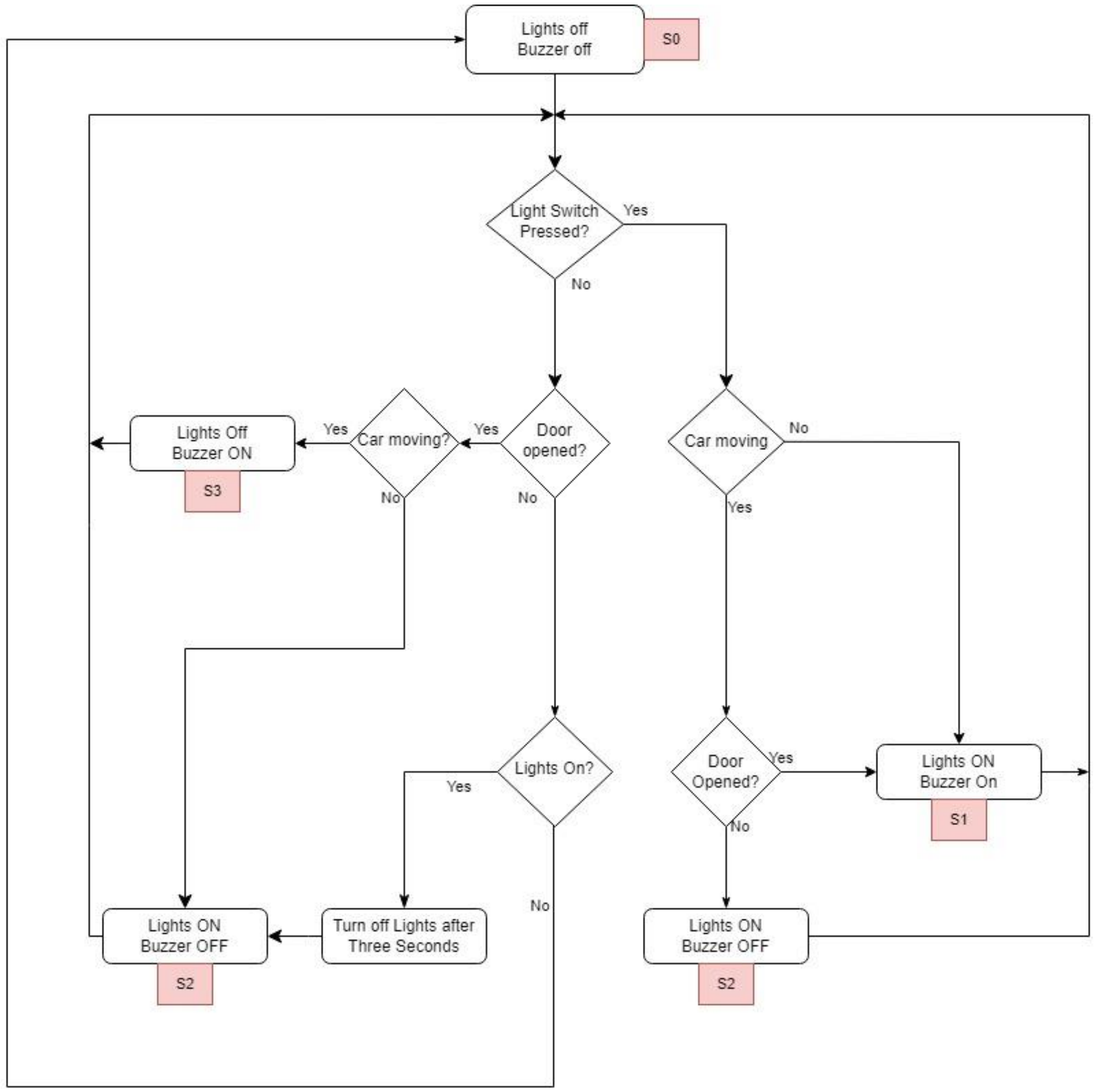
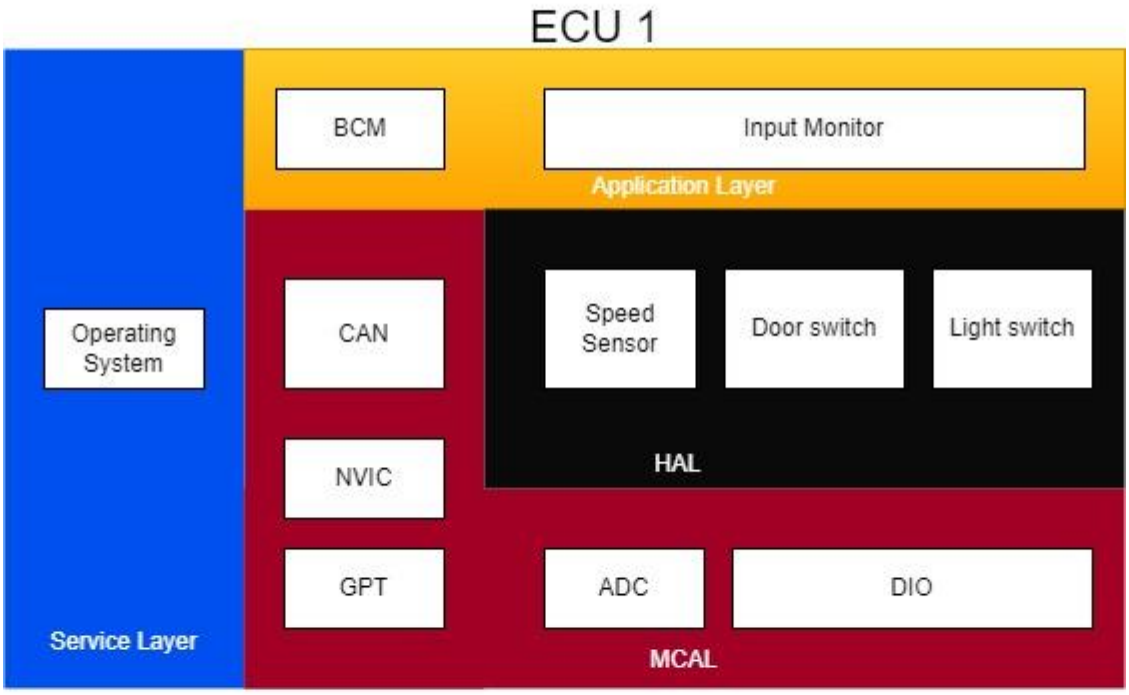
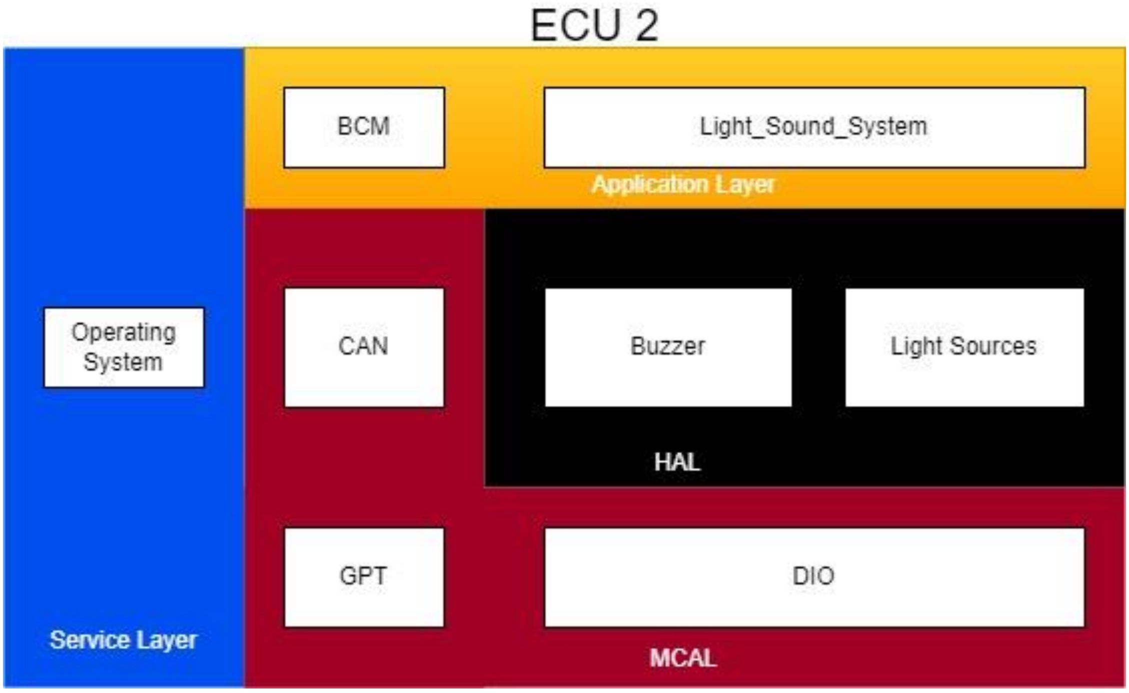
Block Diagram of the requirements



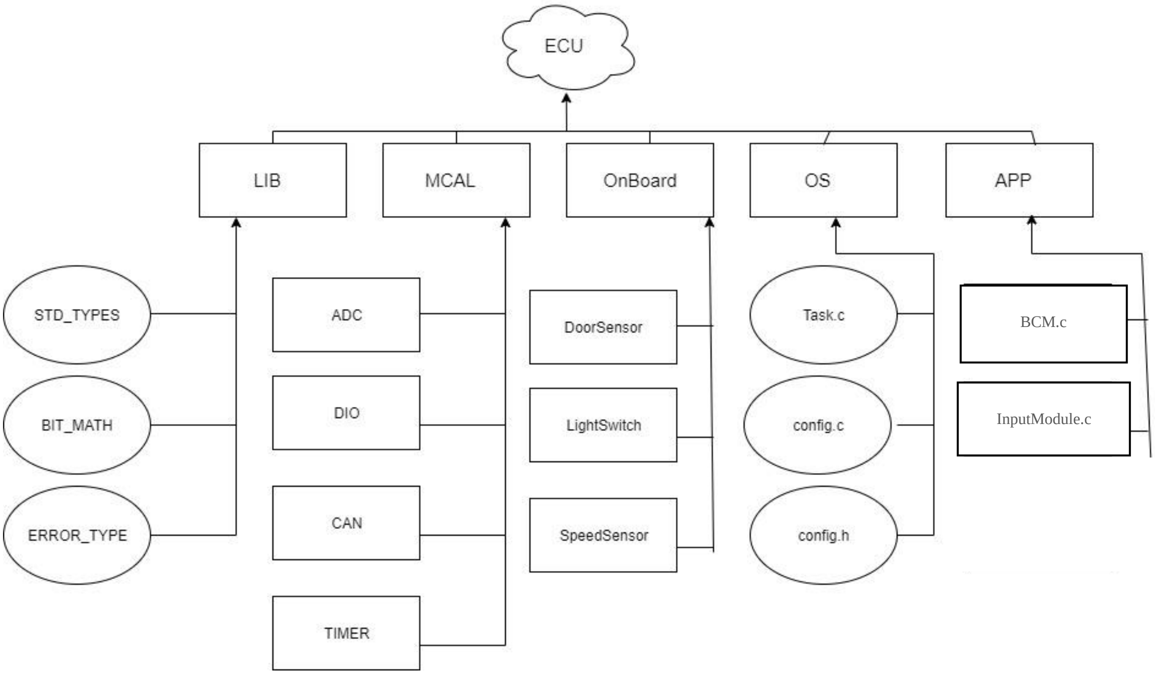
Layered architecture



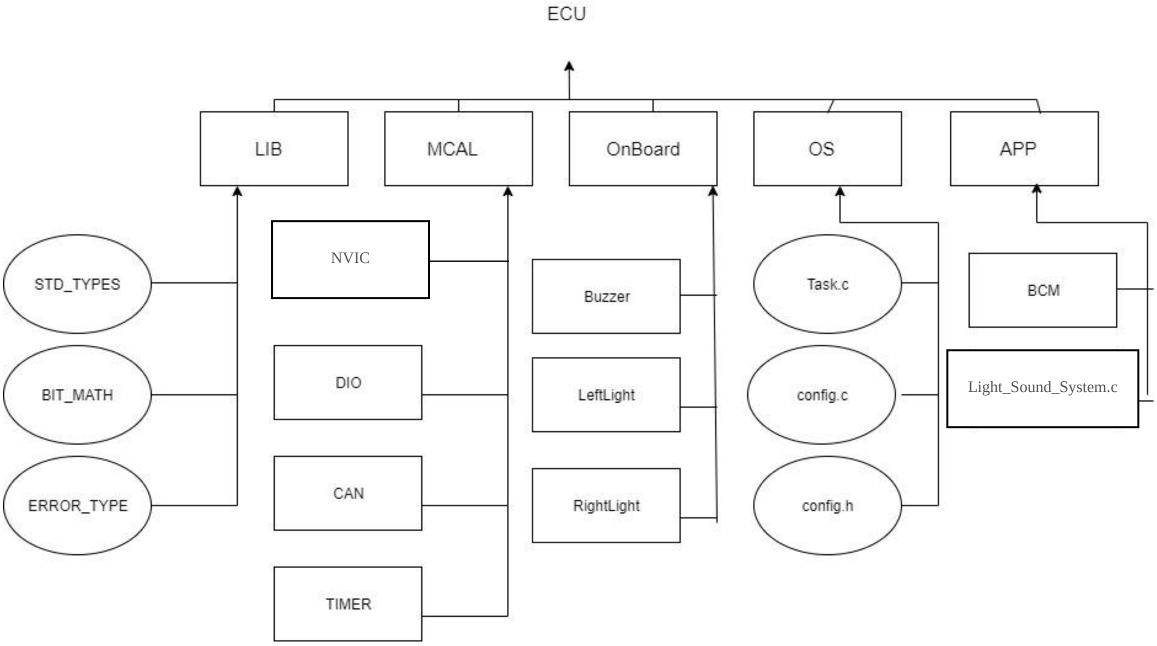


File Structure

ECU1:



ECU2:



Application Layer:

Basic Communication Module(BCM):

API	Description
void BCM_init ()	initialize CAN module and set its configuration according to its config file
Bus_State BCM_IsReady()	Returns the state of CAN bus (ready or not)
Buffer_Status BCM_newData()	Finds if CAN Buffer contains new data
void BCM_updateCANMotionStatus (Motion_State)	Sends the Moving status on the CAN bus to ECU2
void BCM_updateCANDoorStatus (Door_State)	Sends the Door status on the CAN bus to ECU2
void BCM_updateCANLightStatus (LS_State)	Sends the Light Switch status on the CAN bus to ECU2
BCM_action BCM_receiveCANAction ()	returns action that was sent through CAN bus by ECU1
TypeDefs	**Description**
enum Bus_State	contains either: ➢ 0: Bus busy ➢ 1: Bus ready
enum Bus_State	contains either: ➢ 0: Buffer doesn't contain new data ➢ 1: Buffer contains new data

enum BCM_action	Contains received actions: ➢ 0: light switch not pressed ➢ 1: light switch pressed ➢ 2: car stopped ➢ 3: car moving ➢ 4: door closed ➢ 5: door opened

Input Monitor(ECU1):

API	Description
void initSpeedSensor()	Initializes the Speed sensor in Motion detection mode and the sensor’s pin according to config file.
void initLightSwitchMonitor()	Initializes the Light switch pin as input according to config file.
void initDoorMonitor()	Initializes the Door monitor pin as input according to config file.
Motion_State getMotionSensorReading()	Returns the Motion status of the car detected by the speed sensor.
LS_State getLightSwitchState()	Returns the state of the light switch
Door_State getDoorState()	Returns the state of the door wither opened or closed.
TypeDefs	**Description**
enum LS_State	contains either: ➢ 0: Light switch not pressed. ➢ 1: Light switch pressed.
enum Motion_State	contains either ➢ 2: Stopped moving. ➢ 3: Is moving
enum Door_State	contains either: ➢ 4: Door closed ➢ 5: Door opened

Light_Sound_System(LSS): (ECU2)

API	Description
void LSS_init()	Initializes state & action as zeros
void LSS_update_state(LSS_action)	Determines the next state depending on current state and received actions
void LSS_update_action(uint8_t)	Updates the array of actions LSS_Actions that contain Light switch and Door sensor and motion sensor previous states.
LSS_Change LSS_has_state_Changed()	Determines if state has changed since this function was last called.
LSS_DisableLight LSS_LightProtocol()	Checks if Lights are On when they're not supposed to be using LSS_actions and isLightOn()
TypeDefs	**Description**
enum LSS_State	Range (0:3) contains possible States
Uint8_t LSS_action	Array of size 3 contains LS_State, Motion_State and Door_State
enum LSS_DisableLight	contains either: ➢ 0: default ➢ 1: Should Disable Light
LSS_Change	contains either: ➢ 0: No change ➢ 1: State changed

Service Layer:

Operating System:

API	Description
void vTask_periodicSendMotionState()	calls BCM_update_Speed_status() each 5ms to send Motion Status periodically which is obtained by calling getMotionSensorReading().
void vTask_periodicSendSwitchState()	calls BCM_update_switch_state() each 20ms to send switch state periodically which is obtained by calling getLightSwitchState() .
void vTask_periodicSendDoorState()	calls BCM_update_door_state each 10ms to send door state periodically which is obtained by calling getDoorState().
void vTask_periodicReadCANBuffer()	Receives data stored on CAN bus using BCM_receiveCANMotionStatus() Sends the returned data through LSS_update_action(action) Is called every 5ms
vTask_periodicDisableLights()	Disables Lights if they were on when they're not supposed to be. Is called every 10ms

Hardware Abstraction Layer:

Door Switch Driver:(ECU1)

API	Description
void initDoorMonitor (Lock_Pin, Lock_Port)	Initialize pin connected to Door Lock sensor
Door_State isDoorOpened()	Indicates if door is opened.

Speed Sensor Driver:(ECU1)

API	Description
void SpeedSensor_init(SS_Channel)	initialize speed sensor pin.
Motion_State isCarMoving()	indicates if the car is moving.
TypeDefs	**Description**
enum SS_Channel	Determine which ADC channel speed sensor is connected to

Light Switch Driver:(ECU1)

API	Description
void initSwitchMonitor (LS_Pin, LS_PORT)	initialize pin connected to light switch as input
LS_State isLightSwitchPressed()	Indicates if light switch is pressed

Buzzer Driver: (ECU2)

API	Description
void initBuzzer (Buz_Pin, Buz_PORT)	initialize pin connected to Buzzer as output
void ChangeBuzzerState(uint8_t)	Turns the Buzzer on or off
TypeDefs	**Description**
enum Buz _PORT	Range(0:3) for ports A,B,C,D
enum Buz _Pin	Range(0:7) for corresponding pins

Light Sources Driver :(ECU2)

API	Description
void initLightSource (LS_pin, LS_port, isLeft)	Initializes pin connected to light source.
void ChangeLightsState(uint8_t)	Turns lights on or off
Uint8_t isLightsOn(void)	Returns if both light sources are on
TypeDefs	**Description**
enum isLeft	contains either: ➢ 0 for Right. ➢ 1 for Left.

Microcontroller Abstraction layer:

Digital Input Output (DIO):

API	Description
void DIO_Init (IO_Port, IO_Pin, IO_Direction)	Initializes a pin in a specific port as Input or Output
void DIO_Pull(IO_Pull_Direction)	Specifies if the pins will be pulled up or down or float.
Level DIO_ReadPin(IO_Port, IO_Pin)	Returns the input level of a specific pin & port
Level DIO_WritePin(IO_Port, IO_Pin, IO_Level)	Puts a High or Low voltage on the specified pin & port
TypeDefs	**Description**
enum IO_PORT	Range(0,3) corresponding to A,B,C,D
enum IO_PIN	Range(0,7) corresponding to pins in a port
enum IO_LEVEL	contains either: ➢ 0 : Input Low ➢ 1 : Input High.
enum IO_Direction	contains either: ➢ 0 : indicates Input Direction. ➢ 1 : indicates Output Direction.
enum IO_Pull_Direction	Range(0:2) ➢ 0 : pin float ➢ 1 :.pull up ➢ 2 : pull down

General Purpose Time (GPT):

API	Description
void TIMER_init(Timer_Mode)	Initializes timer in specified Mode
void Timer_SetISRFunction (*void (*timerISRfunArg)())	Sets timerISRfunptr = timerISRfunArg to use ISR_Timer to call a callback function.
void TIMER_Start()	Starts timer ticks
ISR_Timer	Calls the callback function timerISRfunptr pointing at
TypeDefs	**Description**
struct Timer_Mode	Determines timer configurations: ➢ Free running or oneshot ➢ Overflow, CTC, PWM ➢ Polling or Interrupt

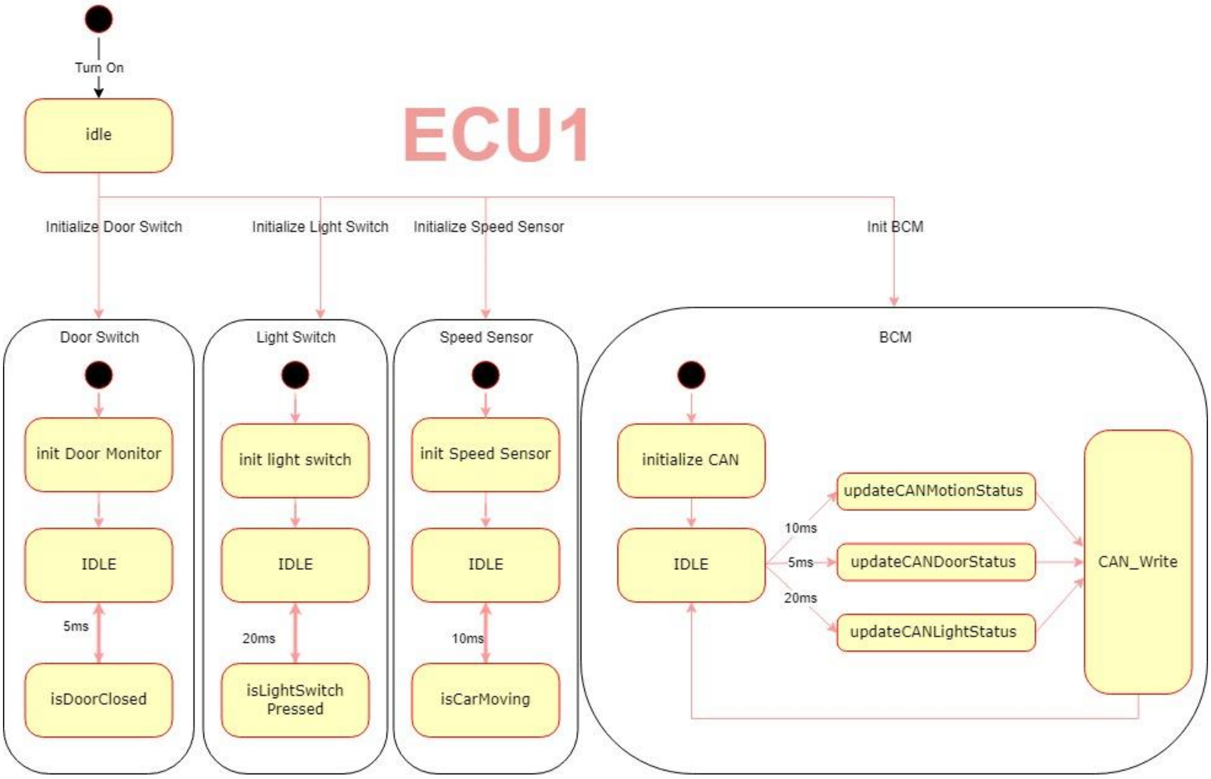
Controller Area Network (CAN) Module:

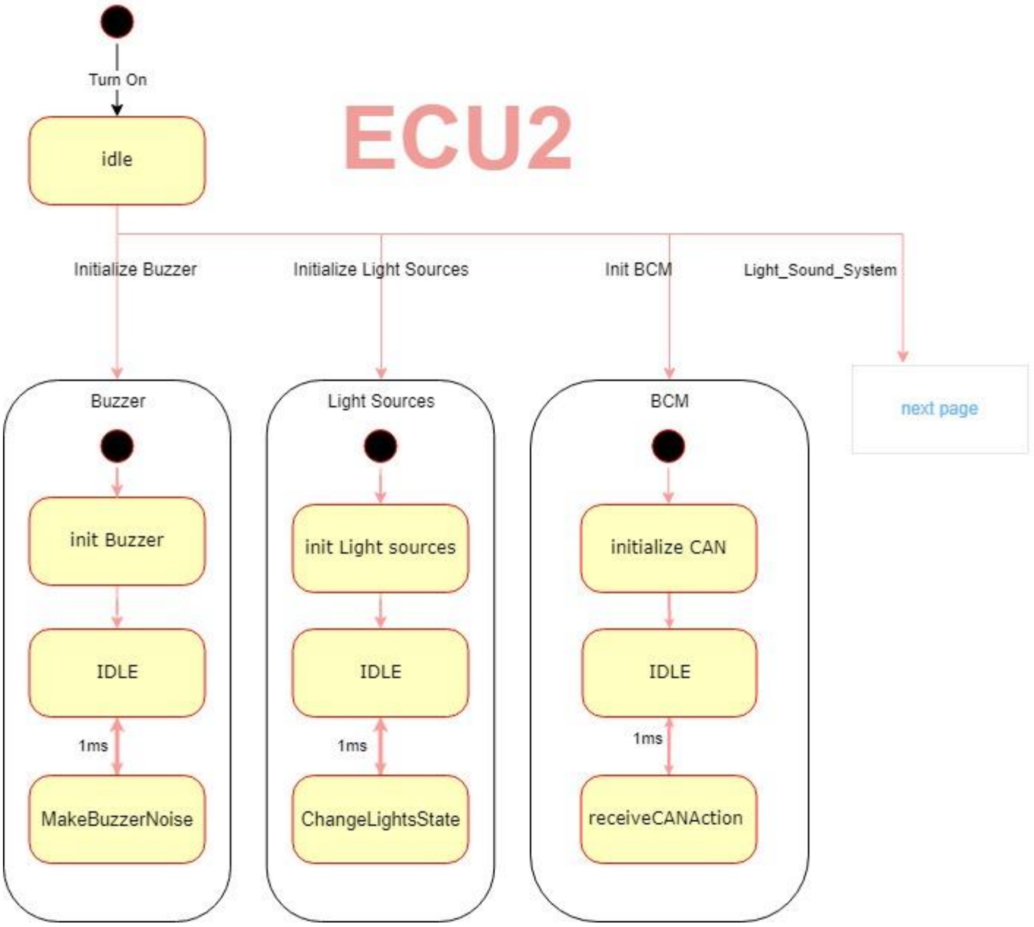
API	Description
void CAN_init (CAN_config)	Initializes CAN module for specific use
void CAN_Write(uint16_t data)	Inserts 2 Bytes of data into the data section in the CAN Frame to be sent on the CAN BUS
Bus_State isCANReady()	Returns the state of the CAN BUS which determines if there’s data on the Bus or not.
uint16_t CAN_Read(void)	Returns stored data in the CAN data Buffer which has been received through the BUS (not used by ECU1)
Buffer_State isRxComplete()	Returns the state of the buffer which determines if Reception has been complete and new data arrived.

Analog to Digital Converter (ADC): (ECU1)

API	Description
void ADC_init(ADC_Channel, Mode)	Initializes ADC with a specific channel and Mode.
uint32_t ADC_StartConversion (ADC_Channel)	Starts conversion on the analog input on ADC_Channel and returns the Digital output
TypeDefs	**Description**
enum ADC_Mode	Range(0,3) Contains one of the ADC four modes.
enum ADC_Channel	Determines ADC channel number

State machine

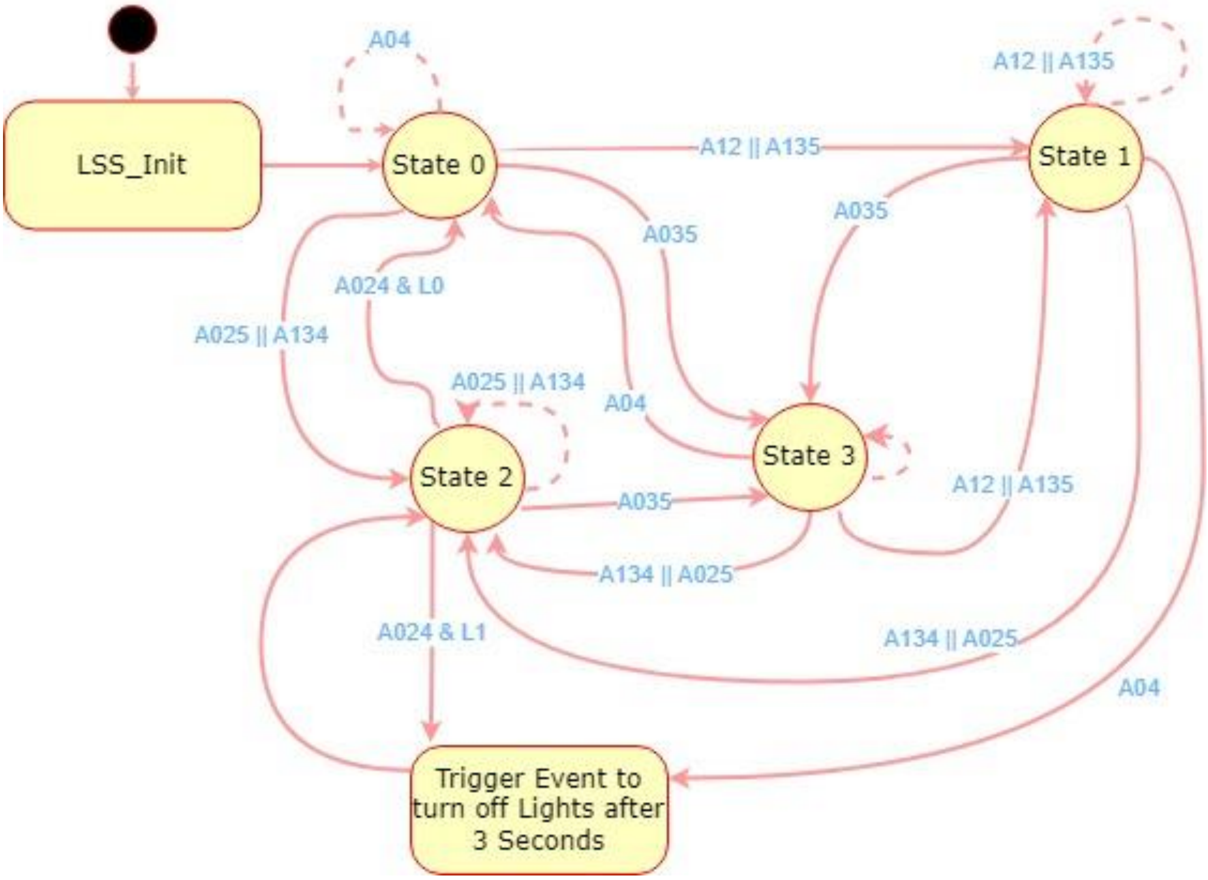




Action	OFF	ON
Light Switch	A0 (released)	A1 (pressed)
Motion Sensor	A2 (Stopped)	A3 (moving)
Door Switch	A4 (Closed)	A5 (Opened)

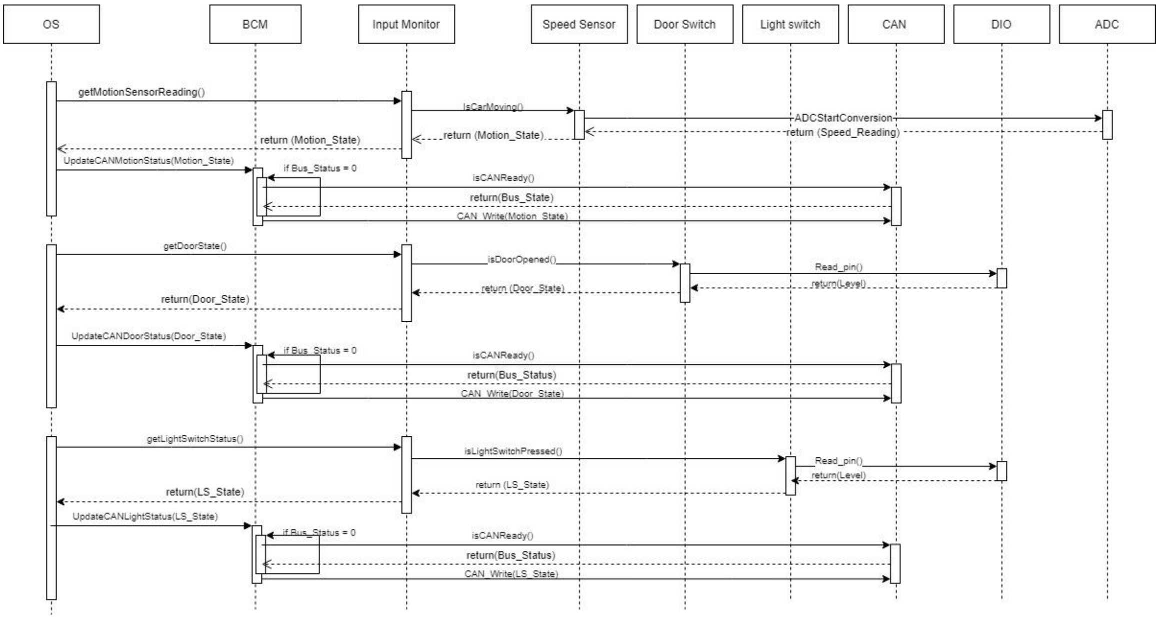
State	0	1	2	3
Light Soruce	OFF	ON	ON	OFF
Buzzer	OFF	ON	OFF	ON

Light_Sound_System

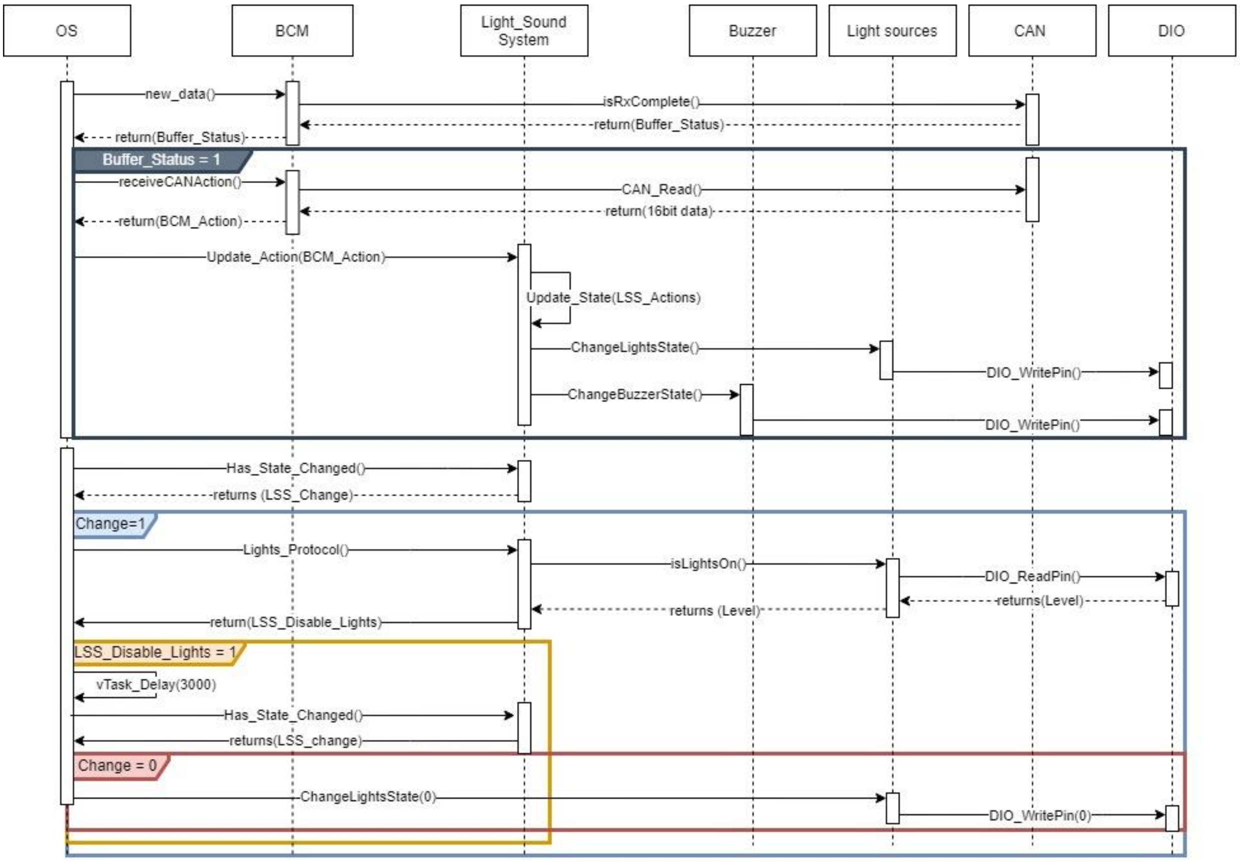


Sequence Diagram

ECU1:



ECU2:



CPU LOAD

CPU Load = (Total Execution Time / Hyper Period) * 100

Assume that each Task executes in around 2ms

For ECU1:

CPU Load = [(4*E1+2*E2+E3)/(20ms)] * 100%

CPU Load = 14/20 * 100 = 70%

Number_Of_Messages per 1s= (4 + 2 + 1)/20 * 1000 = 350

Each message is sent in a frame of 106bits (Standard CAN)

BusLoad per 1second = used capacity / maximum capacity * 100%

used capacity = total bits = frame_size * number_of_messages = 37100

maximum capacity = CAN_Frequency = assume 125KHZ

BusLoad = 37100 / 125000 * 100% = 29.68%

For ECU2:

CPU Load = [(2E1 + E2) / (10ms)] * 100%

Assuming each task is around 2ms

CPU Load = 5/10 = 50%