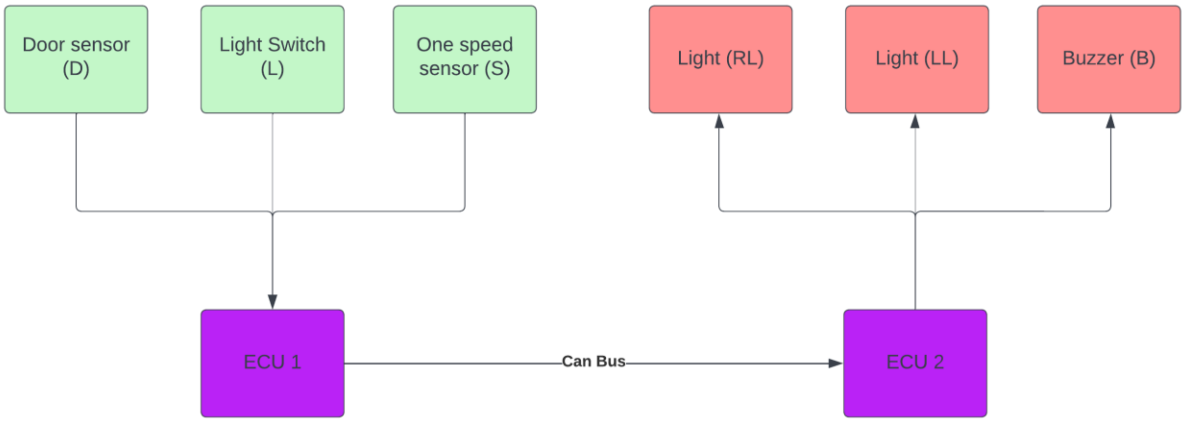
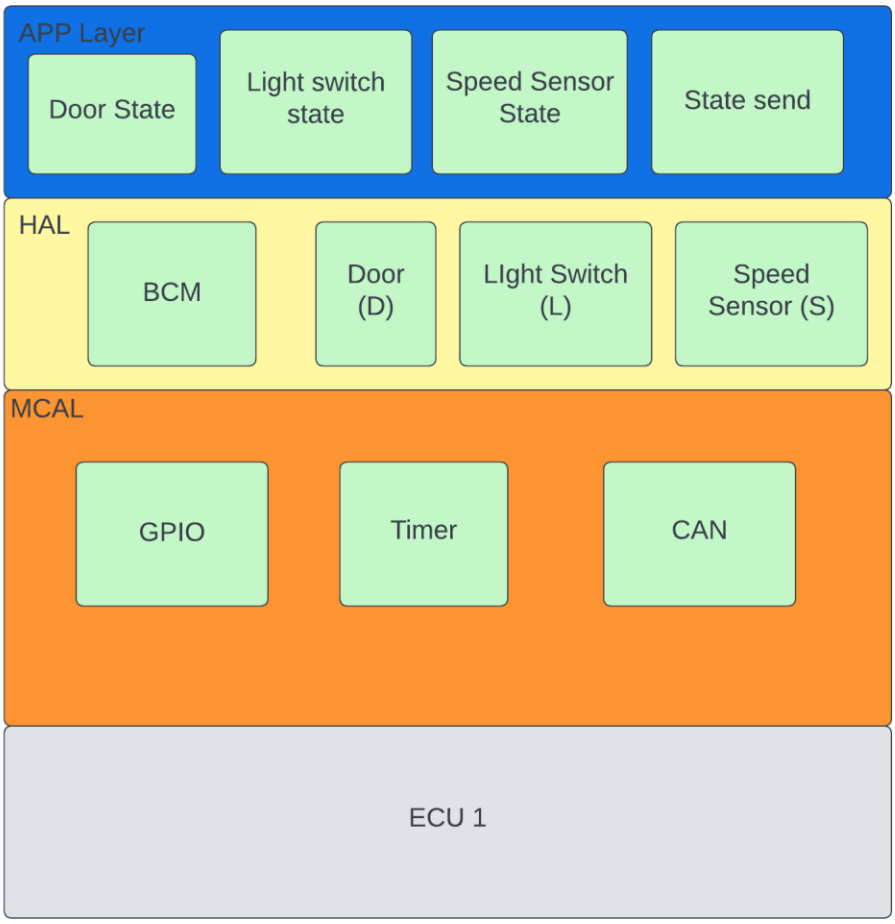
the system schematic (Block Diagram):



1. Static Design Analysis
First ECU1:

1. Make the layered architecture



2. Specify ECU components and modules
    A. MCAL layer :
        I. GPIO module
        II. Timer Module
        III. CAN Module
    B. HAL layer:
        I. BCM module
        II. Door Module
        III. Light Switch Module
        IV. Speed Sensor

C. App Layer:

I. Door State Module

II. Light Switch state Module

III. Speed Sensor State Module

IV. Send state Module

3. List of Api Functions For ECU 1:

First: MCAL layer

1. Module: Timer (GPT timer)

A. Timer_init(Void)

Name	Timer_init
Sync/Async	Synchronus
Reentrancy	Non-reentrant
Parameters in	None
Parameters out	None
Return Value	None
Description	Initializes the timer

B. Timer_GetTimeElapsed(Timer_ChannelType Channel)

Name	Timer_GetTimeElapsed
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Channel Identtfier of the channel in question
Parameters out	None
Return Value	Timer_ValueType //Elapsed time in number of ticks
Description	Gets the time elapsed

C. Timer_GetTimeRemaining(Timer_ChannelType Channel)

Name	Timer_GetTimeRemaining
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Channel //Identifier of the channel in question
Parameters out	None
Return Value	Timer_ValueType //Remaining time in number of ticks
Description	Returns the Remaining time until the set time

D. Timer_StartTimer(Timer_ChannelType Channe, Timer_ValueType Time)

Name	Timer_StartTimer
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Channel //Identifier of the channel in question Time // the time set for this timer to count to
Parameters out	None
Return Value	none
Description	Starts a timer channel

E. Timer_StopTimer(Timer_ChannelType Channe, Timer_ValueType Time)

Name	Timer_StopTimer
Sync/Async	Synchronous
Reentrancy	Reentrant

Parameters in	Channel //Identifier of the channel in question
Parameters out	None
Return Value	none
Description	stops a timer channel

F. Timer_EnableNotification(Timer_ChannelType Channel)

Name	Timer_EnableNotificattion
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Channel //Identifier of the channel in question
Parameters out	None
Return Value	none
Description	Enables interrupt for the channel

G. Timer_DisableNotification(Timer_ChannelType Channel)

Name	Timer_DisableNotificattion
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Channel //Identifier of the channel in question
Parameters out	None
Return Value	None
Description	Disables interrupt for the channel

H. TypeDefs

I. Timer_ValueType:
typedef uint32_t Ttimer_ValueType;
this type simply stores an integer

II. Timer_ChannelType
Typedef enum{
T1 = T1PR,
T2 = T2PR,
Etc:
} timer_ChannelType;
This enum types stores the identifier for the Channel like its name

2. GPIO (and DIO)

A. GPIO_init(Void)

Name	GPIO_init
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters in	None
Parameters out	None
Return Value	None
Description	Initiates the GPIO port

B. Dio_WriteChannel(Dio_port Port , uint8_t PinNumber,Dio_LevelType level)

Name	Dio_WriteChannel
Sync/Async	Synchronous
Reentrancy	Reentrant

Parameters in	Port //port identfiers; PinNumber //the number of the pin; Level //the level to write (High or Low)
Parameters out	None
Return Value	None
Description	Writes in an output pin

C. Void Dio_ReadChannel(Dio_port Port , uint8_t PinNumber,Dio)

Name	Dio_ReadChannel
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Port //port identfiers; PinNumber //the number of the pin;
Parameters out	None
Return Value	Dio_LevelType // returns the level of the pin whether it is high or low
Description	Reads input

D. Typedefs

Dio_LevelType

typedef enum{

LOW,

HIGH}Dio_LevelType;

A port type that defines high and low

typedef enum{PORTA,PORTB,PORTC,PORTD,PORTE,PORTF,} Dio_port;

The types gives an identifier to all ports on the system1`

3. CAN

A. Can_Init(void)

Name	Can_Init(void)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters in	None
Parameters out	None
Return Value	None
Description	Initiates the CAN module

B. Can_tx(Can_Channel_Num Channel, uint8_t* Data)

Name	Can_tx
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Channel // an identifier for the channel number Data // a pointer to an 8 bit data to transmit
Parameters out	None
Return Value	Std_ReturnType // E_ok or E_NOT_OK
Description	Transmit through the CAN bus

C. Can_Rx(Can_Channel_Num Channel)

Name	Can_Rx
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Channel // an identifier for the channel number
Parameters out	None
Return Value	Std_ReturnType // E_ok or E_NOT_OK
Description	Recieve through the CAN bus

D. Typedefine:
Can_Channel_Num

typedef uint32_t Can_Channel_Num;

Second: HAL Layer

1. BCM

A. BCM_init(void)

Name	BCM_Init(void)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters in	None
Parameters out	None
Return Value	None
Description	Initiates the BCM module, from external config, like which can Channel to use etc.

B. BCM_Send(uint8_t* Data)

Name	BCM_Send
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Data // a pointer to an 8 bit data to transmit
Parameters out	None
Return Value	Std_ReturnType // E_ok or E_NOT_OK
Description	Transmit through the Data through the CAN bus

C. BCM_Recieve(void);

Name	BCM_Recieve
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	None
Parameters out	None
Return Value	Std_ReturnType // E_ok or E_NOT_OK
Description	Recieve through the CAN bus

2. Door module
   A. Door_Init(void)

Name	Light_Switch_Init(void)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters in	None
Parameters out	None
Return Value	None
Description	Initiates the Door module, from external config, like which GPIO Channel to use etc.

   B. Door_Status(Void)

Name	Door_Status
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	None
Parameters out	None
Return Value	Uint8_t // retuens the state as a one or a zero
Description	Checks if the door is open if it is it returns 1 if not it returns zero

3. Light Switch Module
   A. Light_Switch_Init(void)

Name	Light_Switch_Init(void)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters in	None
Parameters out	None
Return Value	None
Description	Initiates the Light Switch module, from external config, like which GPIO Channel to use etc.

   B. Light_Switch_Status(Void)

Name	Light_Switch_Status
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	None
Parameters out	None
Return Value	Uint8_t // retuens the state as a one or a zero
Description	Checks if the light Switch is Closed if it is it returns 1 if not it returns zero

4. Speed_Sensor module:
   A. Speed_Sensor_Init(void)

Name	Speed_Sensor_Init(void)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters in	None
Parameters out	None
Return Value	None
Description	Initiates the Speed Sensor module, from external config, like which GPIO Channel to use etc.

B. Speed_Sensor_Status(Void)

Name	Speed_Sensor_Status
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	None
Parameters out	None
Return Value	Uint8_t // retuens the state as a one or a zero
Description	Checks if the Speed Sensor is 1(car moving), 0 (car not moving)

Third: App Layer:

1. Door State module
   A. Get_Door_Status(void)

Name	Get_Door_Status
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	None
Parameters out	None
Return Value	Uint8_t // If the Door is open returns 0xF1 if Closed Returns 0xF0
Description	returns the status with an identifier so the other Ecu Can analyses it after receiving it through the Can bus

2. Light Switch State Module
   A. Get_Light_Switch_Status(void)

Name	Get_Light_Switch_Status
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	None
Parameters out	None
Return Value	Uint8_t // If the light switch is open returns 0xE1 if Closed Returns 0xE0
Description	returns the status with an identifier so the other Ecu Can analyses it after receiving it through the Can bus

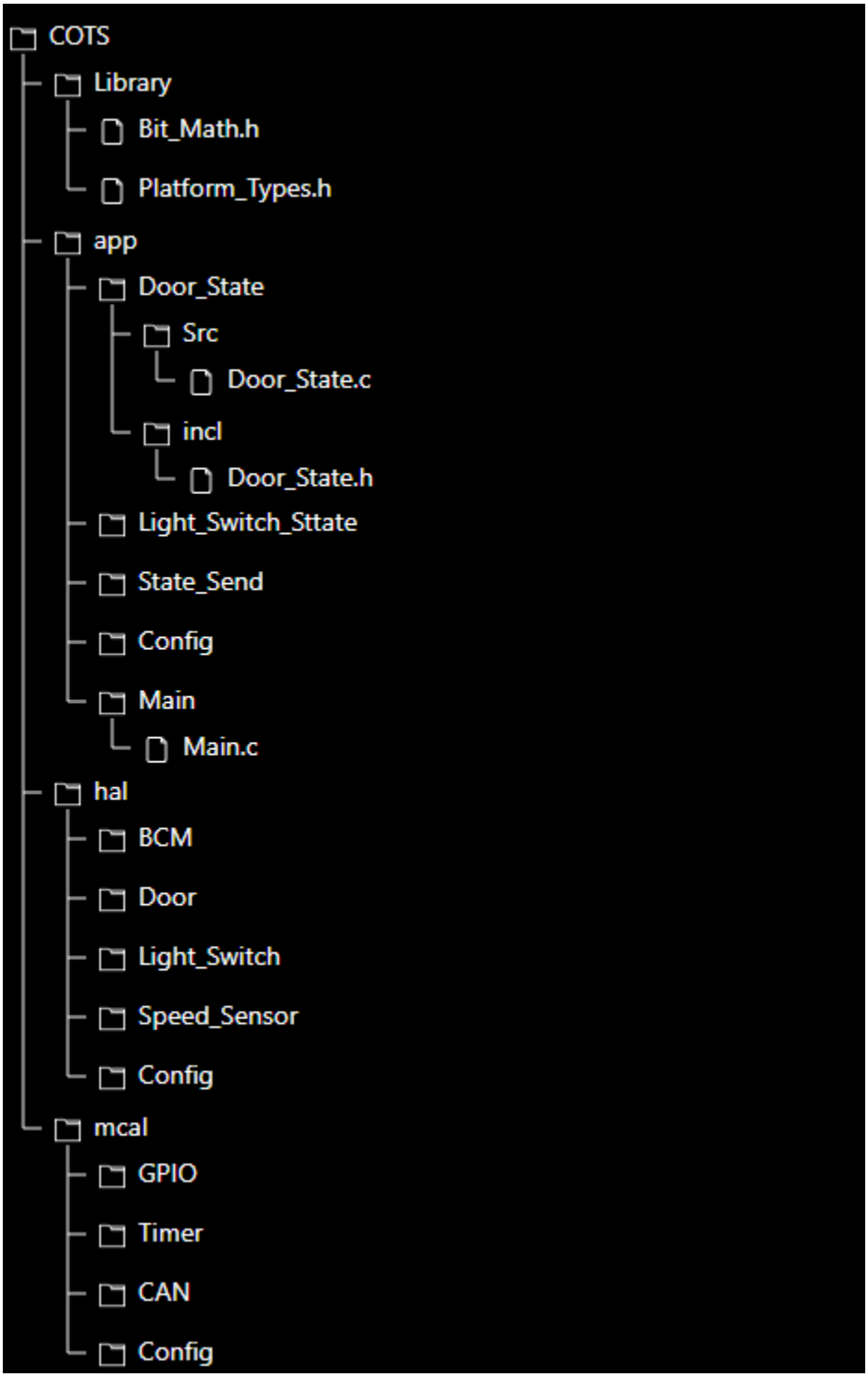
3. Speed Sensors status module
   A. Get_Speed_Sensor_Status(void)

Name	Get_Speed_Sensor_Status
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	None
Parameters out	None
Return Value	Uint8_t // If the Car is moving returns 0xD1 if not Returns 0xD0
Description	returns the status with an identifier so the other Ecu Can analyses it after receiving it through the Can bus

4. State Send module(uint8_t* Status)

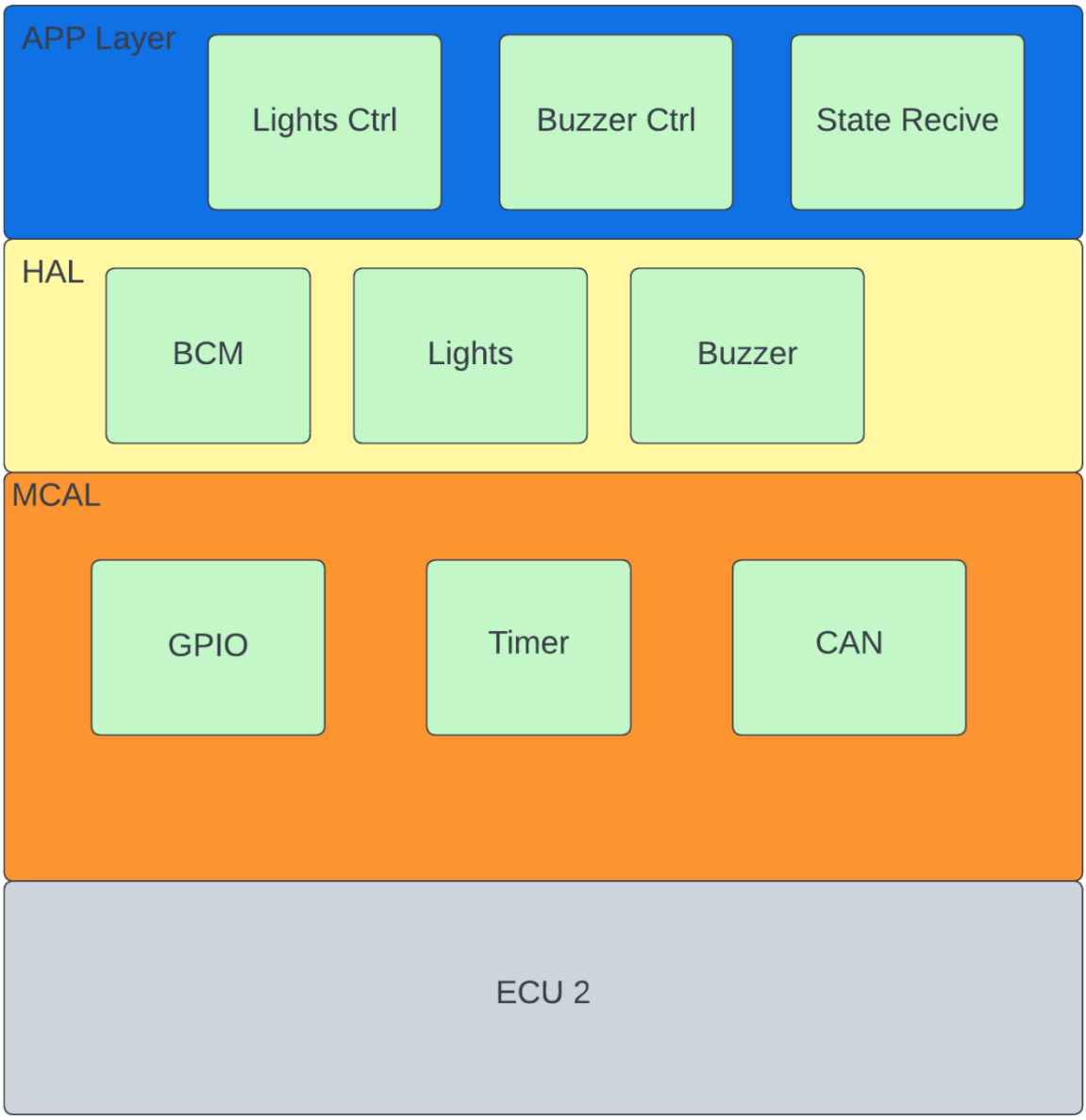
Name	Get_Speed_Sensor_Status
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Status // a pointer to the variable with the status to send
Return Value	Uint8_t // If the Car is moving returns 0xD1 if not Returns 0xD0
Description	Sends the status from previous modules to through the CAN bus

4. Prepare your folder structure according to the previous points



Second ECU2:

1. Make the layered architecture



2. Specify ECU components and modules
    1. MCAL layer :
        1. GPIO module
        2. Timer Module
        3. CAN Module
    2. HAL layer:
        1. BCM Module
        2. Lights Module
        3. Buzzer Module
    3. App layer
        1. Light Ctrl
        2. Buzzer Ctrl
        3. State Receive
3. Provide full detailed APIs for each module as well as a detailed description for the used typedefs

First: MCAL layer

1. Module: Timer (GPT timer)

A. Timer_init(Void)

Name	Timer_init
Sync/Async	Synchronus
Reentrancy	Non-reentrant
Parameters in	None
Parameters out	None
Return Value	None
Description	Initializes the timer

B. Timer_GetTimeElapsed(Timer_ChannelType Channel)

Name	Timer_GetTimeElapsed
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Channel Identtfier of the channel in question
Parameters out	None
Return Value	Timer_ValueType //Elapsed time in number of ticks
Description	Gets the time elapsed

C. Timer_GetTimeRemaining(Timer_ChannelType Channel)

Name	Timer_GetTimeRemaining
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Channel //Identifier of the channel in question
Parameters out	None
Return Value	Timer_ValueType //Remaining time in number of ticks
Description	Returns the Remaining time until the set time

D. Timer_StartTimer(Timer_ChannelType Channe, Timer_ValueType Time)

Name	Timer_StartTimer
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Channel //Identifier of the channel in question Time // the time set for this timer to count to
Parameters out	None
Return Value	none
Description	Starts a timer channel

E. Timer_StopTimer(Timer_ChannelType Channe, Timer_ValueType Time)

Name	Timer_StopTimer
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Channel //Identifier of the channel in question
Parameters out	None
Return Value	none
Description	stops a timer channel

F. Timer_EnableNotification(Timer_ChannelType Channel)

Name	Timer_EnableNotificattion
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Channel //Identifier of the channel in question
Parameters out	None
Return Value	none
Description	Enables interrupt for the channel

G. Timer_DisableNotification(Timer_ChannelType Channel)

Name	Timer_DisableNotificattion
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Channel //Identifier of the channel in question
Parameters out	None
Return Value	None
Description	Disables interrupt for the channel

H. TypeDefs

III. Timer_ValueType:
typedef uint32_t Ttimer_ValueType;
this type simply stores an integer

IV. Timer_ChannelType
Typedef enum{
T1 = T1PR,
T2 = T2PR,
Etc:
} timer_ChannelType;
This enum types stores the identifier for the Channel like its name

2. GPIO (and DIO)
   A. GPIO_init(Void)

Name	GPIO_init
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters in	None
Parameters out	None
Return Value	None
Description	Initiates the GPIO port

   B. Dio_WriteChannel(Dio_port Port , uint8_t PinNumber,Dio_LevelType level)

Name	Dio_WriteChannel
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Port //port identfiers; PinNumber //the number of the pin; Level //the level to write (High or Low)
Parameters out	None
Return Value	None
Description	Writes in an output pin

   C. Void Dio_ReadChannel(Dio_port Port , uint8_t PinNumber,Dio)

Name	Dio_ReadChannel
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Port //port identfiers; PinNumber //the number of the pin;
Parameters out	None
Return Value	Dio_LevelType // returns the level of the pin whether it is high or low
Description	Reads input

   D. Typedefs
      Dio_LevelType
      typedef enum{
          LOW,
          HIGH}Dio_LevelType;
      A port type that defines high and low

      typedef enum{PORTA,PORTB,PORTC,PORTD,PORTE,PORTF,} Dio_port;
      The types gives an identifier to all ports on the system1`
3. CAN
   A. Can_Init(void)

Name	Can_Init(void)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters in	None
Parameters out	None
Return Value	None

Description	Initiates the CAN module

B. Can_tx(Can_Channel_Num Channel, uint8_t* Data)

Name	Can_tx
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Channel // an identifier for the channel number Data // a pointer to an 8 bit data to transmit
Parameters out	None
Return Value	Std_ReturnType // E_ok or E_NOT_OK
Description	Transmit through the CAN bus

C. Can_Rx(Can_Channel_Num Channel)

Name	Can_Rx
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Channel // an identifier for the channel number
Parameters out	None
Return Value	Std_ReturnType // E_ok or E_NOT_OK
Description	Recieve through the CAN bus

D. Typedefine:
Can_Channel_Num

typedef uint32_t Can_Channel_Num;

Second: Hal Layer

1. BCM

A. BCM_init(void)

Name	BCM_Init(void)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters in	None
Parameters out	None
Return Value	None
Description	Initiates the BCM module, from external config, like which can Channel to use etc.

B. BCM_Send(uint8_t* Data)

Name	BCM_Send
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Data // a pointer to an 8 bit data to transmit
Parameters out	None
Return Value	Std_ReturnType // E_ok or E_NOT_OK
Description	Transmit through the Data through the CAN bus

C. BCM_Recieve(void);

Name	BCM_Recieve
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	None
Parameters out	None
Return Value	Std_ReturnType // E_ok or E_NOT_OK
Description	Recieve through the CAN bus

2. Lights Module
   a. Lights_init()

Name	Lights_Init(void)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters in	None
Parameters out	None
Return Value	None
Description	Initiates the Lights module, from external config, like which GPIO Channel to use for each light etc.

   b. Light_Switch(Light_id_t ID, Lights_mode_t Mode)

Name	Light_Switch
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	ID // an idetfier for which Light Left or right Mode // Whether Lights_on , or off
Parameters out	None
Return Value	None
Description	Switches the state of the lights

```
typedef enum{
    left,
    Right} Light_id;
```

This type defines the left and right lights/

```
typedef enum{
    Ligth_on,
    Light_off} Lights_mode_t;
```

This type defines the state of the light whether on/off

3. Buzzer Module
   a. Buzzer_init()

Name	Buzzer_Init(void)
Sync/Async	Synchronous
Reentrancy	Non-Reentrant
Parameters in	None
Parameters out	None
Return Value	None
Description	Initiates the Buzzer module, from external config, like which GPIO Channel to use etc.

b. Buzzer_Switch(Buzzer_mode_t Mode)

Name	Buzzer_Switch
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Mode // Whether Buzzer_on , or off
Parameters out	None
Return Value	None
Description	Switches the state of the Buzzer

typedef enum{
Buzzer_on,
Buzzer_off} Buzzer_mode_t;

This type defines the state of the light whether on/off

Third : App Layer

1. Lights Ctrl module
   A. Lights_Ctrl(Lights_Mode_t Mode)

Name	Lights_Ctrl
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Mode // Whether Light_on , or off
Parameters out	None
Return Value	None
Description	Switches the state of both lights

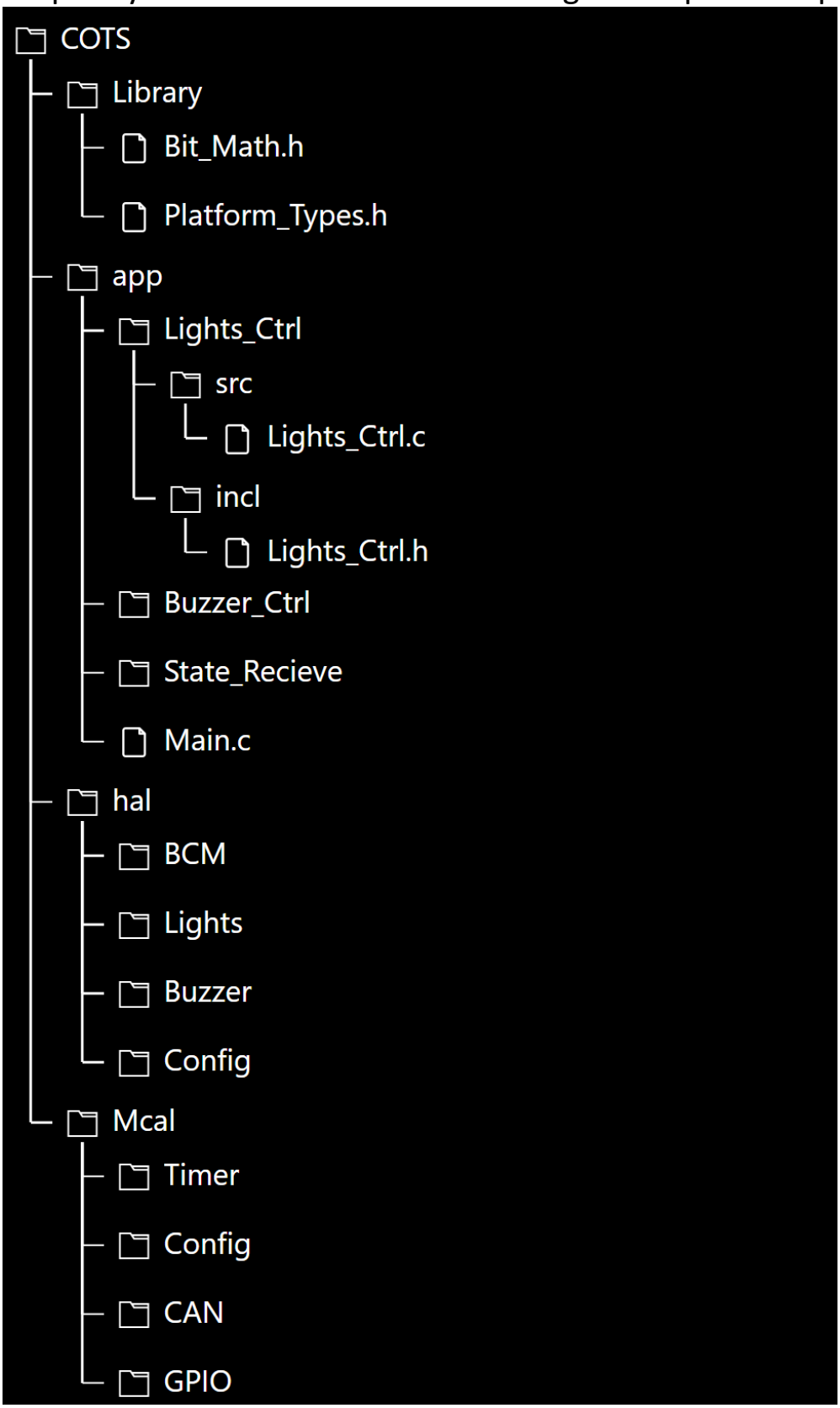
2. Buzzer Ctrl Module
   A. Buzzer_Ctrl(Buzzer_Mode_t Mode)

Name	Lights_Ctrl
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	Mode // Whether Buzzer_on , or off
Parameters out	None
Return Value	None
Description	Switches the state of the Buzzer

3. State Receive module
   State_Recieve(Void)

Name	State_Receive
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters in	None
Parameters out	None
Return Value	Uint8_t // an 8 bit variable with the identifier and state received through CAN
Description	Switches the state of the Buzzer

4. Prepare your folder structure according to the previous points



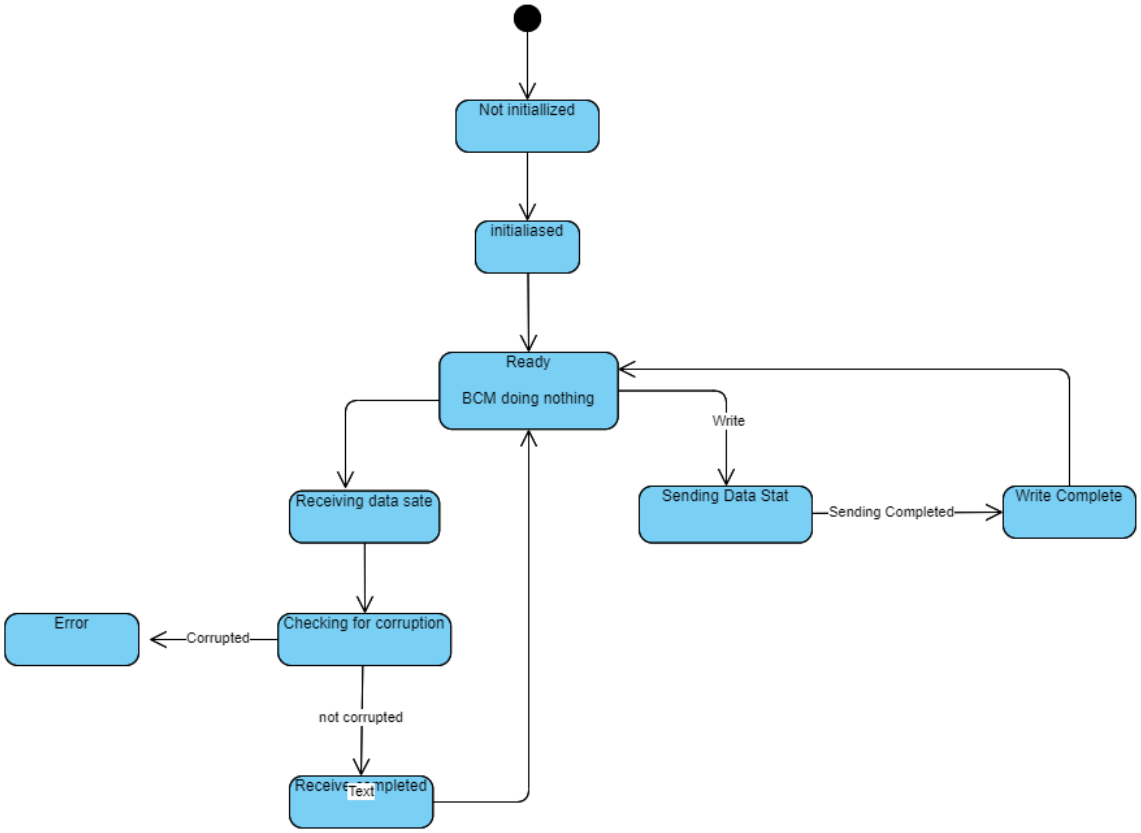
Dynamic Design

I. For ECU 1

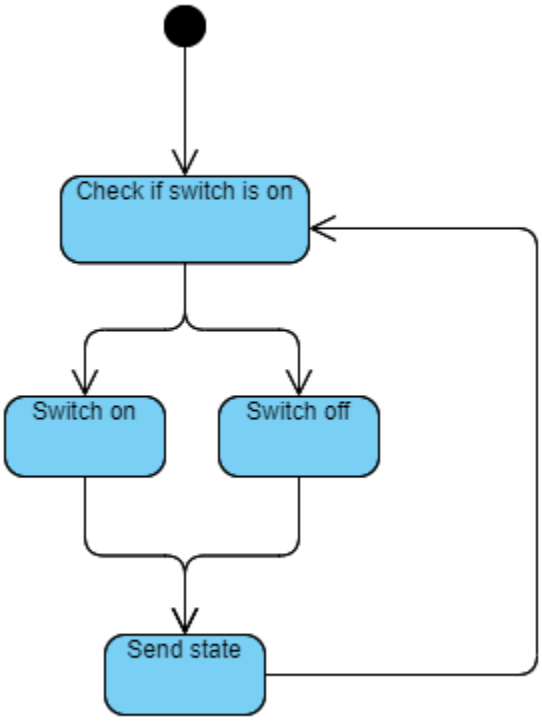
1. Draw a state machine diagram for each ECU component

A. OS Component

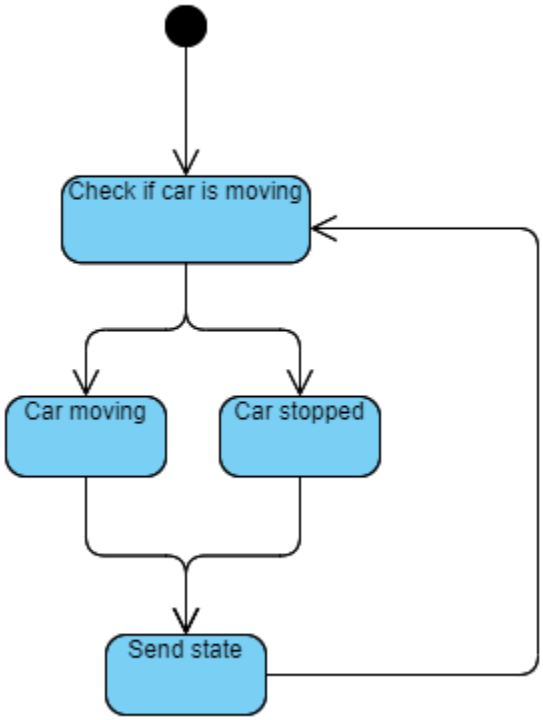
a. BCM



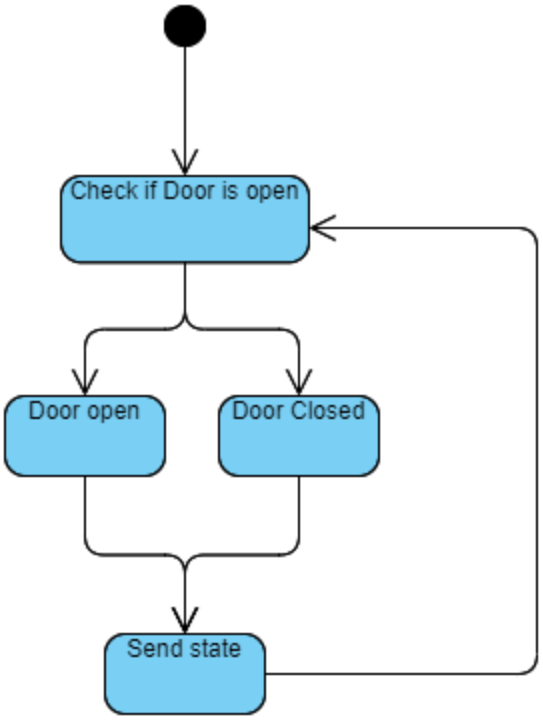
B. Software component
   a. Switch button state



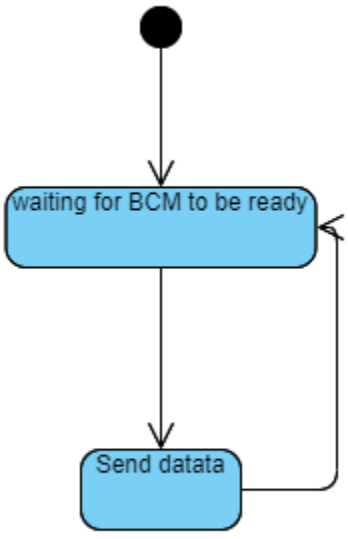
   b. Car state



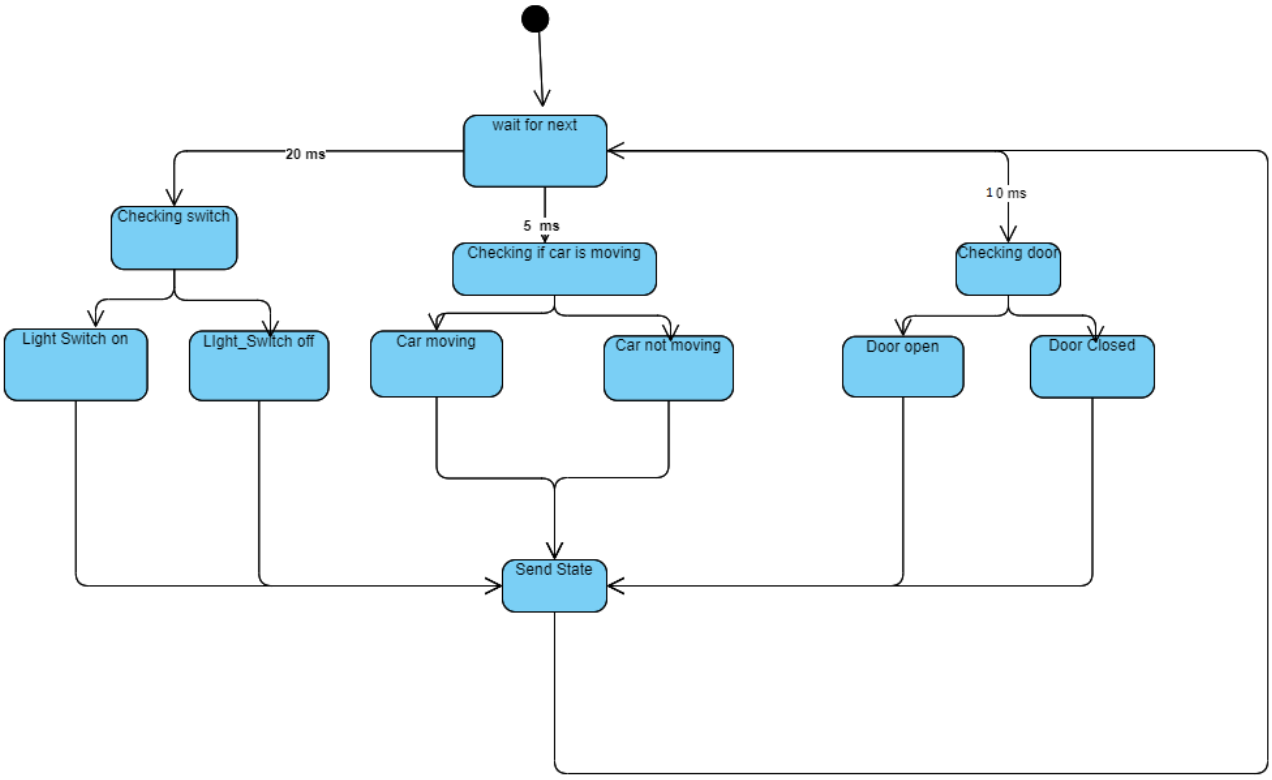
c. Door state



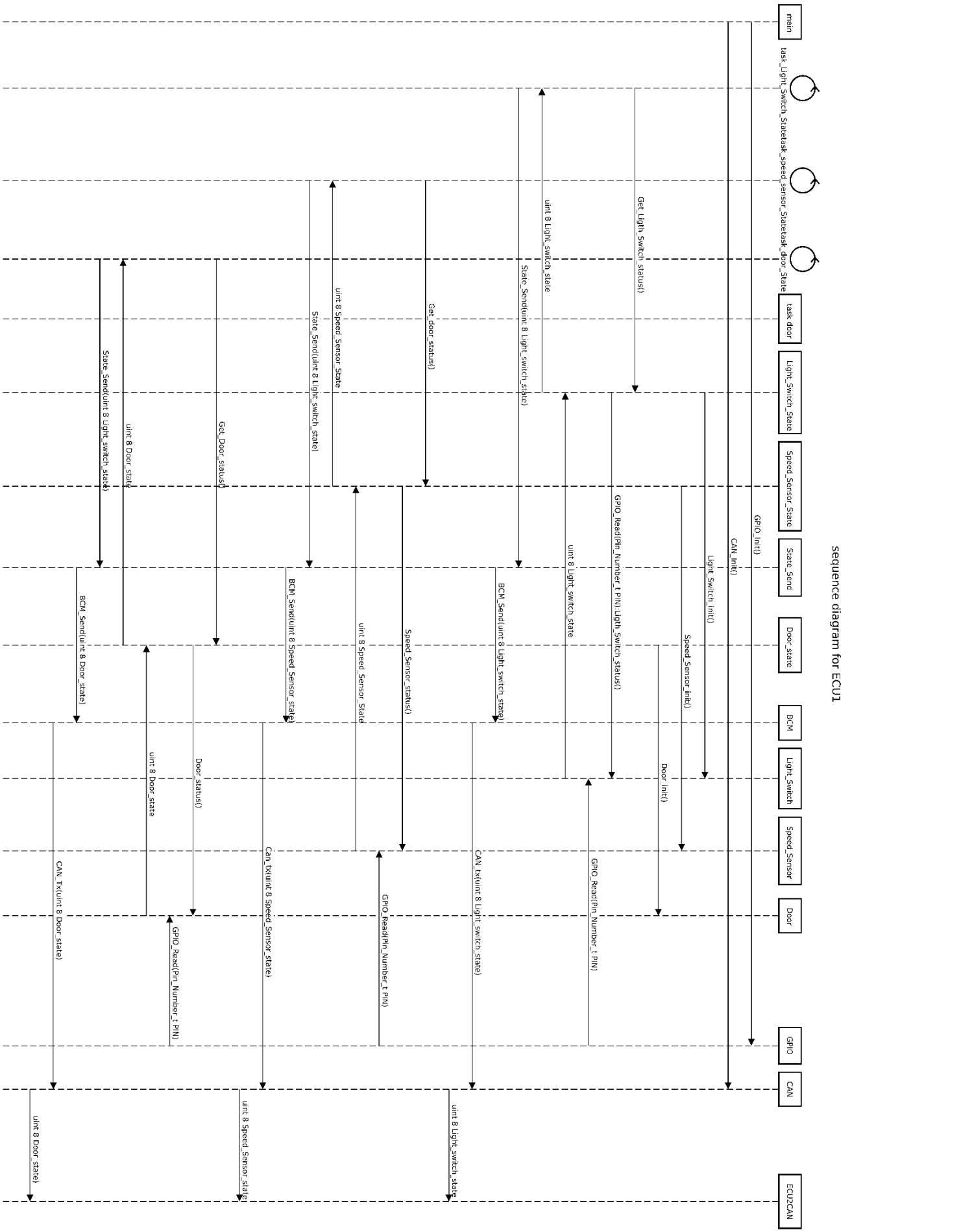
d. State send



2. Draw a state machine diagram for the ECU operation



3. Draw the sequence diagram for the ECU



4. Calculate CPU load for the ECU 1
   Note: no code was written and it is all just theoretical calculations

Hyper period will be 20 ms in my design

It will consist of 3 tasks of periods 5,10,20

Assuming each task is similar as they do basically the same thing and saying the entire process takes around 350 μs to execute and another 150 μs for debounce and any other safety related features.

So Execution time will be 500 μs for each task

So at 20 ms

The 5 ms periodicity task would have run four times for a total time of 2 ms

The 10 ms periodicity task would have run two times for a total time of 1 ms

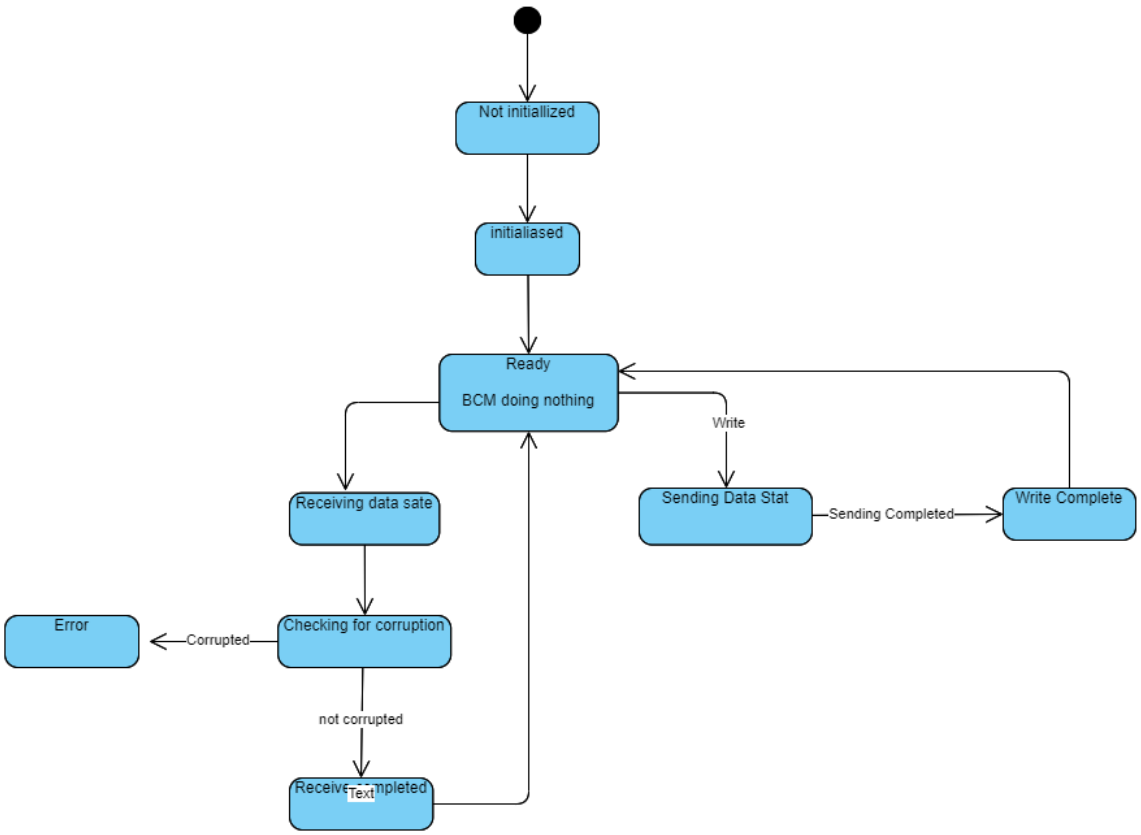
The 20 ms periodicity task would have run once for a total time of 0.5 ms

Then the total execution time is 1.75 ms

$$ECU\ 1\ load = \frac{Excecution\ time\ per\ hyper\ period}{hyper\ period} = \frac{3.5}{20} = 17.5\%$$

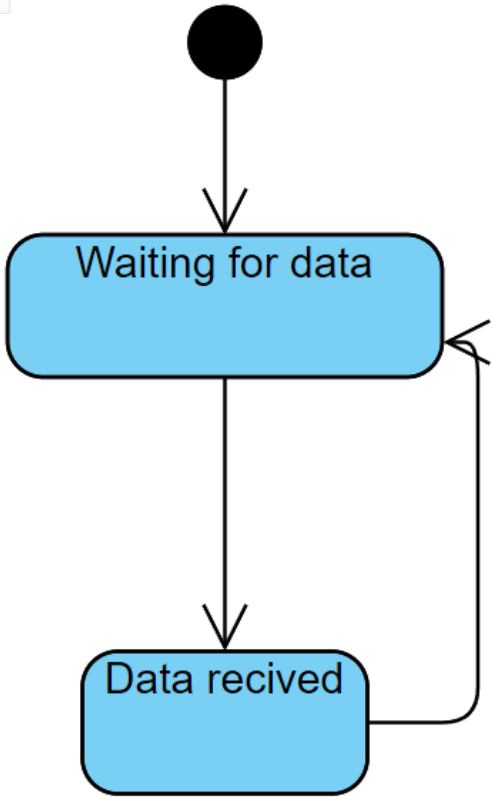
II. For ECU 2

1. Draw a state machine diagram for each ECU component
   A. OS Component
      a. BCM

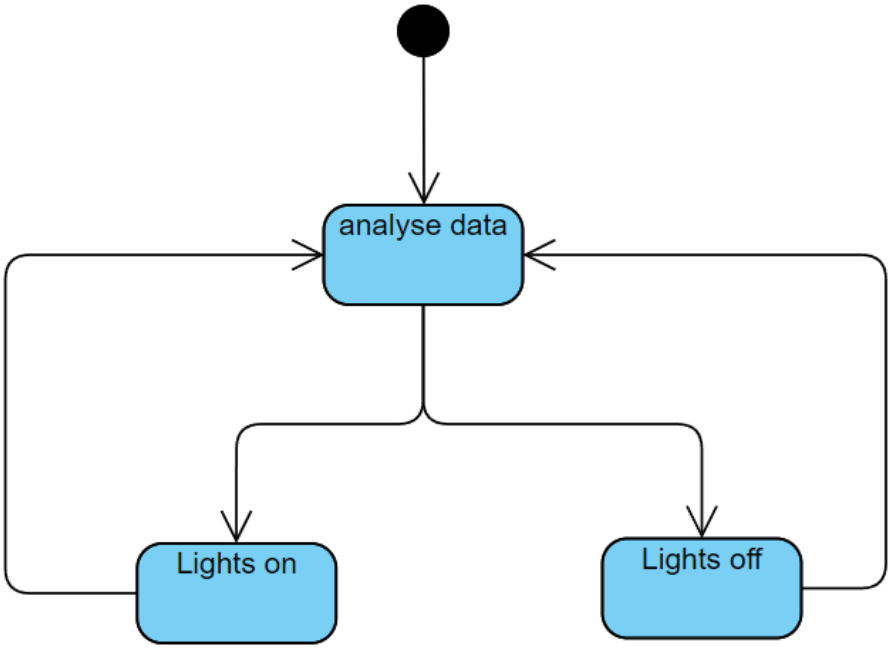


B. Software Components

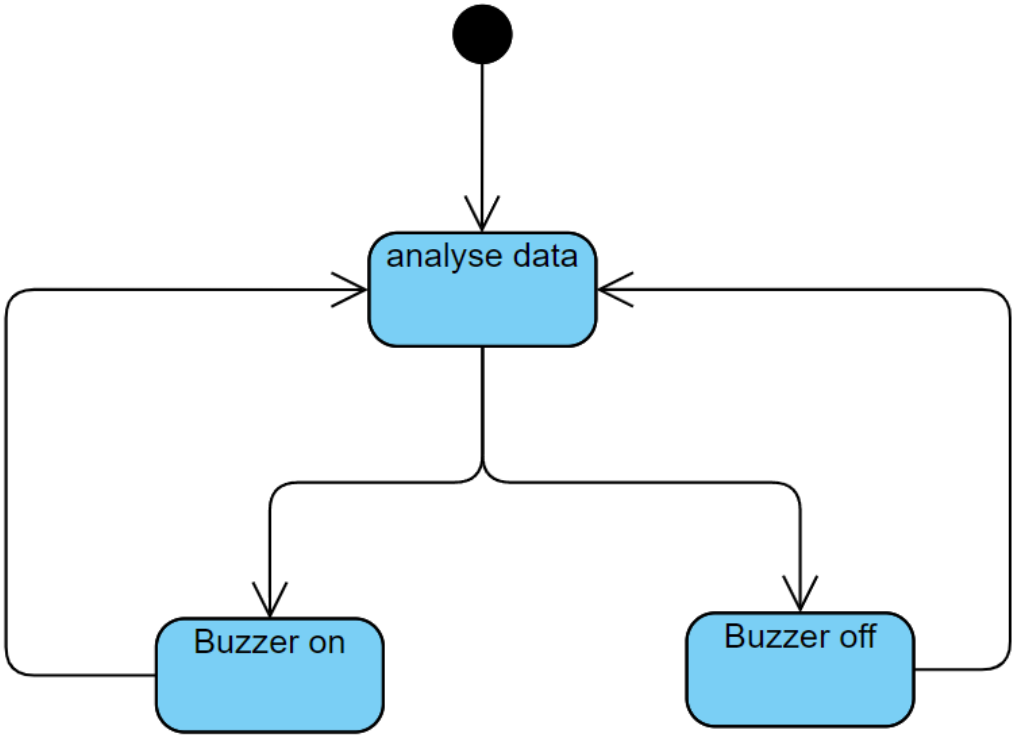
a. State send



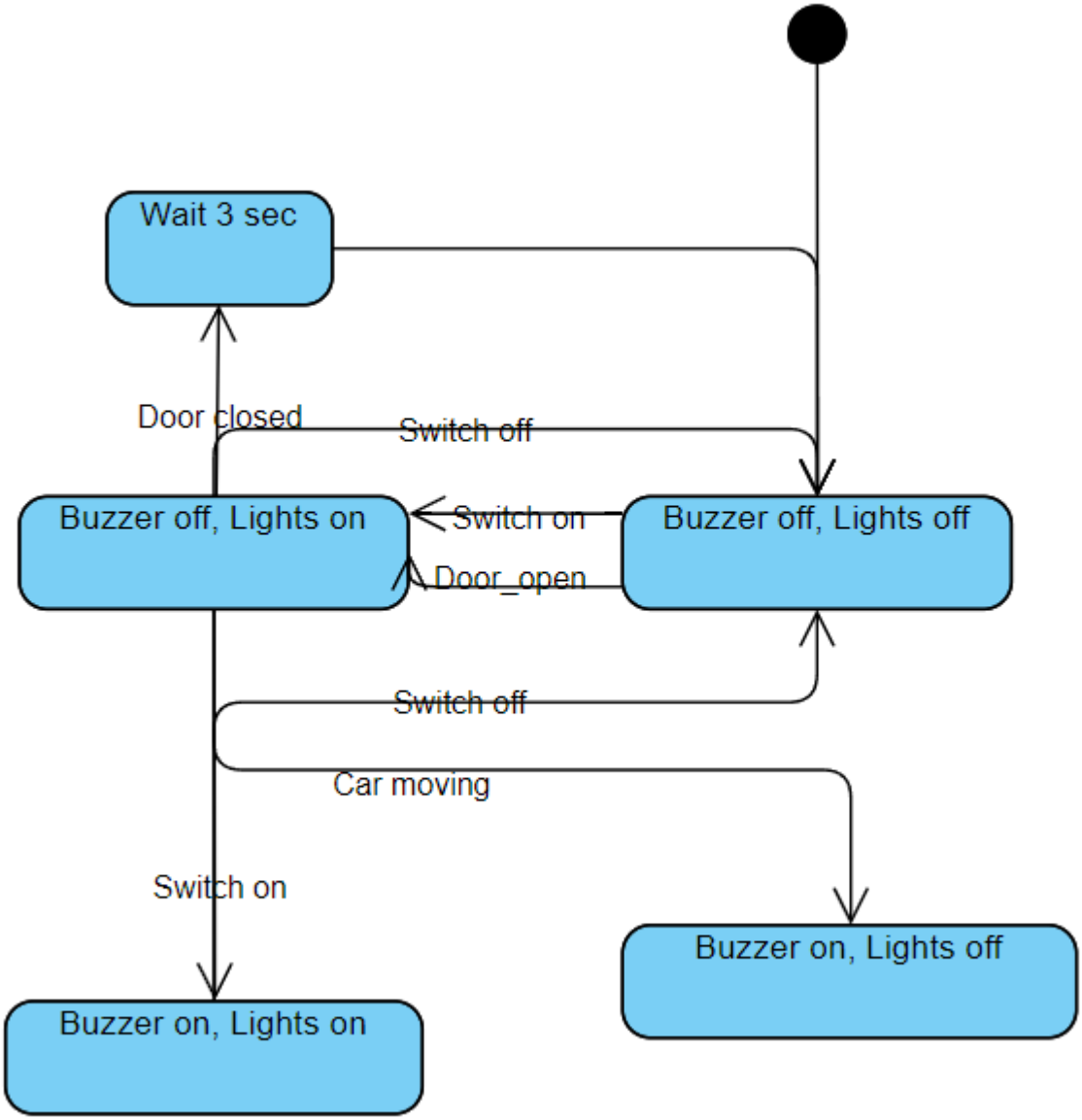
b. Lights Ctrl



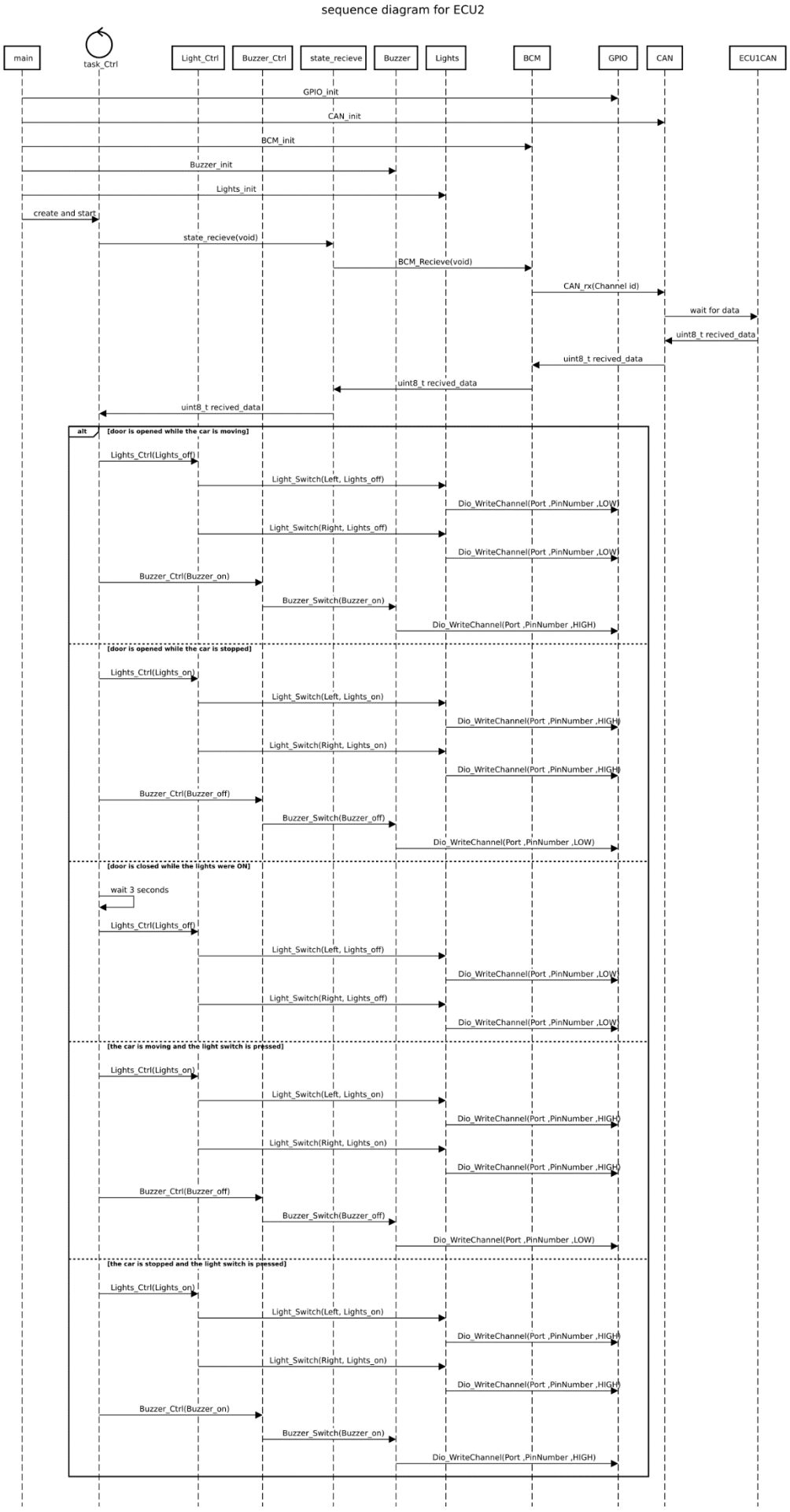
c. Buzzer Ctrl



2. Draw a state machine diagram for the ECU operation



3. Draw the sequence diagram for the ECU



4. Calculate CPU load for the ECU 2

Note: no code was written and it is all just theoretical calculations

Here it is only one task with 5 ms periodicity this task checks everything then controls

Assuming this task takes 400 μs to receive and Ctrl using the data from ECU 2

Then execution time will be 400 μs

$$ECU\ 1\ load = \frac{Excecution\ time\ per\ hyper\ period}{hyper\ period} = \frac{.4}{5} = 8\%$$

III. Calculate bus load in your system

Assuming single wire CAN interface

As single wire CAN wire has a rate of 33.3 kbit/s and we are sending a single byte at once

So, it takes around 300 μs for each operation we do exactly 7 CAN transmissions in 20ms

So, the average CAN bus load= $\frac{300\times7}{20000} = 10.5\%$