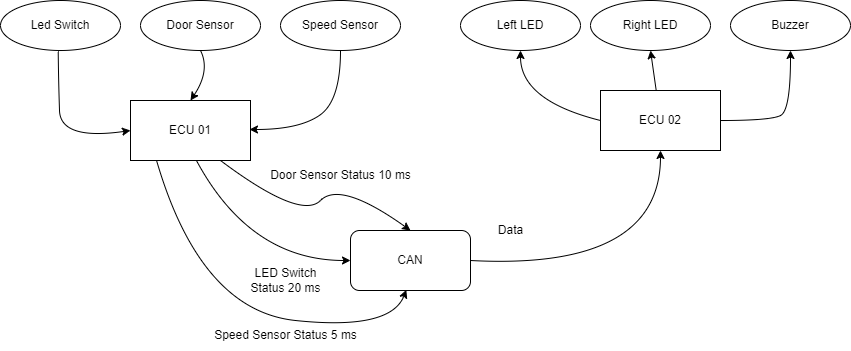
Automotive Door Control System Design

Email: [msmb.mail@gmail.com](mailto:msmb.mail@gmail.com)

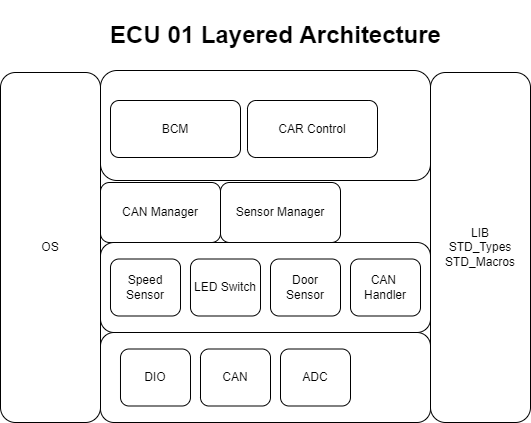
Name: Michael Samir

Static Design:

Block Diagram:



ECU 1 Layered architecture



APIs and typedefs

DIO Driver:

typedef unsigned char u8

typedef struct DIO\_ConfigType

1. ➢ void DIO\_Init(const DIO\_ConfigType \* ConfigPtr, u8 size)

Name: DIO\_Init

Arguments:

* o Name: ConfigPtr
* o Type: pointer to DIO\_ConfigType
* o Range: structure size
* o Description: pointer to array that has all configurations to the selected pins passed by use (ex: pin number, type, speed…)

------------------------------------------------------------------

* o Name: size
* o Type: u8
* o Range: 0:10
* o Description: argument that has size of array of used pins

------------------------------------------------------------------

Return type: void

Description: This API called to configure GPIO pins in the ECU using array of struct => typedef struct DIO\_ConfigType;

1. ➢ u8 DIO\_ReadChannel(u8 ChannelId)

Name: DIO\_ReadChannel

Arguments:

* o Name: Channel Id
* o Type: u8
* o Range: 0:10
* o Description: Channel number to be read

Return type: u8

Description: API to read the value of GPIO Channel.

Name: DIO\_WriteChannel

Arguments:

* o Name: Channel Id
* o Type: u8
* o Range: 0:10
* o Description: Channel number to be written

------------------------------------------------------------------

* o Name: Value
* o Type: u8
* o Range: 0:1
* o Description: Value to be written

------------------------------------------------------------------

Return type: void

Description: API to write the value of GPIO Channel.

1. ➢ u8 DIO\_ReadPort (u8 PortId)

Name: DIO\_ReadPort

Arguments:

* o Name: PortId
* o Type: u8
* o Range: 0:10
* o Description: Port to be read

Return type: u8

Description: API to read the value of GPIO Port.

1. ➢ Void DIO\_WritePort (u8 PortId, u8 Value)

Name: DIO\_WritePort

Arguments:

* o Name: PortId
* o Type: u8
* o Range: 0:10
* o Description: Port to be written

------------------------------------------------------------------

* o Name: Value
* o Type: u8
* o Range: 0:1
* o Description: Value to be written

------------------------------------------------------------------

Return type: void

Description: API to write the value of GPIO Port.

**ADC Driver:**

Name: ADC\_Init

Arguments:

* o Name: channels
* o Type: u8
* o Range: 0:10
* o Description: the channel number to work as ADC

Return type: void

Description: This API called to Initialize the needed GPIO pin as ADC pins

1. ➢ u8 ADC\_ReadChannel (u8 channel)

Name: ADC\_ ReadChannel

Arguments:

* o Name: channel
* o Type: u8
* o Range: 0:10
* o Description: the channel number to work as ADC

Return type: u8 ->the value read by ADC

Description: This API to read Value of ADC channel

**CAN Driver:**

1. ➢ void CAN\_Init(void)

Name: CAN\_Init

Return type: void

Description: API to initializes CAN module.

1. ➢ void CAN\_SetBaudrate (u16 Baudrate)

Name: CAN\_SetBaudrate

Arguments:

* o Name: Baudrate
* o Type: u16
* o Range: 0: 65535
* o Description: the new baud rate

Return type: void

Description: This API to set the baud rate configuration of the CAN controller.

1. ➢ void CAN\_Write (u16 data);

Name: CAN\_Write

Arguments:

* o Name: data
* o Type: u16
* o Range: 0: 65535
* o Description: data would be sent

Return type: void

Description: API to send Data via CAN

Name: CAN\_Read

Return type: u16

Description: Receive data from CAN

**Door Sensor:**

Must include “DIO Driver”

1. ➢ void DoorSensor\_Init (u8 ChannelId)

Name: DoorSensor\_Init

Arguments:

* o Name: Channel
* o Type: u8
* o Range: 0:10
* o Description: Channel connected to Door Sensor

Return type: void

Description: this API to Initialize Channel as Door Sensor

1. ➢ u8 DoorSensor\_Read (u8 ChannelId)

Name: DoorSensor\_Read

Arguments:

* o Name: ChannelId
* o Type: u8
* o Range: 0:10
* o Description: Channel connected to Door Sensor

Return type: u8 ->the State of Door Sensor

Description: this API to Read Channel of GPIO for Door Sensor

Speed Sensor:

Must include “ADC Driver”

1. ➢ Void SpeedSensor\_Init (u8 ChannelId)

Name: SpeedSensor\_Init

Arguments:

* o Name: ChannelId
* o Type: u8
* o Range: 0:10
* o Description: Channel connected to Speed Sensor

Return type: void

Description: This API to initialize Channel of GPIO as Speed Sensor

1. ➢ u8 SpeedSensor\_Read (u8 ChannelID)

Name: SpeedSensor\_Read

Arguments:

* o Name: ChannelId
* o Type: u8
* o Range: 0:10
* o Description: Channel connected to Speed Sensor

Return type: u8 -> value of Speed Sensor

Description: This API to Read the state of Speed Sensor for the specified ADC channel

LED Switch:

Must include “DIO Driver”

Name: Switch\_Init

Arguments:

* o Name: ChannelId
* o Type: u8
* o Range: 0:10
* o Description: Channel connected to LED Switch

Return type: void

Description: This API to initialize Channel of GPIO as LED Switch

Name: Switch\_Read

Arguments:

* o Name: ChannelId
* o Type: u8
* o Range: 0:10
* o Description: Channel connected to LED Switch

Return type: u8-> state of Switch

Description: This API to Read the specified GPIO pin for LED Switch

CAN Handler:

To enable total abstraction, a handler will be added as a point of contact between the CAN manager and the can Protocol.

CAN Manager:

1. o Name: CANManager\_Init
2. o Return type: void
3. o Description: API for initialization of communication(using CAN\_Init())
4. o Name: CANManager\_Send
5. o Return type: void
6. o Description: API for sending messages between layers (using CAN\_Write())

**Sensor Manager:**

1. ➢ void SensorsManager\_Init (void)
2. o Name: SensorsManager\_Init
3. o Return type: void
4. o Description: API to initialization of all sensors by calling (Switch\_Init (u8 ChannelId), DoorSensor\_Init (u8 ChannelId), SpeedSensor\_Init (u8 ChannelId))
5. ➢ Void SensorsManager\_Read(void)
6. o Name: SensorsManager\_Read
7. o Return type: void
8. o Description: API to get sensors readings (DoorSensor\_Read (u8 ChannelId), SpeedSensor\_Read (u8 ChannelId), Switch\_Read (u8 ChannelId))

**BCM:**

1. ➢ void BCM\_Init ()
2. o Name: BCM\_Init
3. o Return type: void
4. o Description: this API call the API in OS to establish CAN connection (CANManager\_Init ())
5. ➢ void BCM\_Send ()
6. o Name: BCM\_ Send
7. o Return type: void
8. o Description: this API call the API in OS to Send the status messages to ECU2 (CANManager\_Send ())

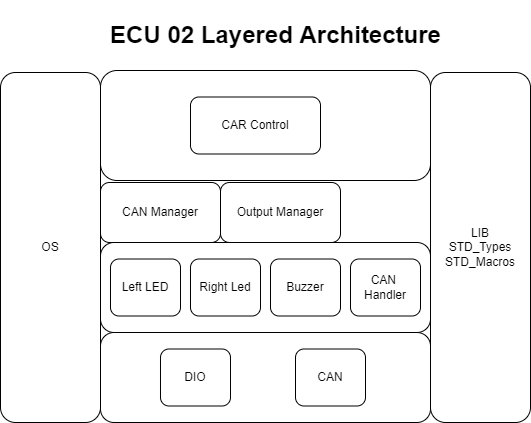
**Car Control:**

1. o Name: InputDevices\_Init
2. o Return type: void

* o Description: API to call the OS API to initialize input devices (SensorsManager\_Init ())

1. o Name: InputDevices\_Control
2. o Return type: void
3. o Description: API to Call OS API to read input devices readings (SensorsManager\_Read ())

ECU 02 Layered Architecture



**DIO Driver:**

typedef unsigned char u8

typedef struct DIO\_ConfigType

1. ➢ void DIO\_Init (const DIO\_ConfigType \* ConfigPtr, u8 size)

Name: DIO\_Init

Arguments:

* o Name: ConfigPtr
* o Type: pointer to DIO\_ConfigType
* o Range: structure size
* o Description: pointer to array that has all configurations to the selected pins passed by use (ex: pin number, type, speed…)

------------------------------------------------------------------

* o Name: size
* o Type: u8
* o Range: 0:10
* o Description: argument that has size of array of used pins

------------------------------------------------------------------

Return type: void

Description: This API called to configure GPIO pins in the ECU using array of struct => typedef struct DIO\_ConfigType;

1. ➢ u8 DIO\_ReadChannel (u8 ChannelId)

Name: DIO\_ReadChannel

Arguments:

* o Name: ChannelId
* o Type: u8
* o Range: 0:10
* o Description: Channel number to be read

Return type: u8

Description: API to read the value of GPIO Channel.

Name: DIO\_WriteChannel

Arguments:

* o Name: ChannelId
* o Type: u8
* o Range: 0:10
* o Description: Channel number to be written

------------------------------------------------------------------

* o Name: Value
* o Type: u8
* o Range: 0:1
* o Description: Value to be written

------------------------------------------------------------------

Return type: void

Description: API to write the value of GPIO Channel.

1. ➢ u8 DIO\_ReadPort (u8 PortId)

Name: DIO\_ReadPort

Arguments:

* o Name: PortId
* o Type: u8
* o Range: 0:10
* o Description: Port to be read

Return type: u8

Description: API to read the value of GPIO Port.

Name: DIO\_WritePort

Arguments:

* o Name: PortId
* o Type: u8
* o Range: 0:10
* o Description: Port to be written

------------------------------------------------------------------

* o Name: Value
* o Type: u8
* o Range: 0:1
* o Description: Value to be written

------------------------------------------------------------------

Return type: void

Description: API to write the value of GPIO Port.

**CAN Driver:**

1. ➢ void CAN\_Init(void)

Name: CAN\_Init

Return type: void

Description: API to initializes CAN module.

1. ➢ void CAN\_SetBaudrate (u16 Baudrate)

Name: CAN\_SetBaudrate

Arguments:

* o Name: Baudrate
* o Type: u16
* o Range: 0: 65535
* o Description: the new baud rate

Return type: void

Description: This API to set the baud rate configuration of the CAN controller.

1. ➢ void CAN\_Write (u16 data);

Name: CAN\_Write

Arguments:

* o Name: data
* o Type: u16
* o Range: 0: 65535
* o Description: data would be sent

Return type: void

Description: API to send Data via CAN

Name: CAN\_Read

Return type: u16

Description: Receive data from CAN

**Right LED Driver:**

1. ➢ void RL\_Init (u8 ChannelId)

Name: RL\_Init

Arguments:

* o Name: ChannelId
* o Type: u8
* o Range: 0:10
* o Description: Channel connected to Right LED

Return type: void

Description: API to Initialize Channel of GPIO as Right LED

1. ➢ void RL\_ON (u8 ChannelId)

Name: RL\_ON

Arguments:

* o Name: ChannelId
* o Type: u8
* o Range: 0:10
* o Description: Channel connected to Right LED

Return type: void

Description: API to make Right LED ON

1. ➢ void RL\_OFF (u8 ChannelId)

Name: RL\_OFF

Arguments:

* o Name: ChannelId
* o Type: u8
* o Range: 0:10
* o Description: Channel of GPIO connected to Right LED

Return type: void

Description: API to make Right LED OFF

**Left LED Driver:**

1. ➢ void LL\_Init (u8 ChannelId)

Name: LL\_Init

Arguments:

* o Name: ChannelId
* o Type: u8
* o Range: 0:10
* o Description: Channel connected to Left LED

Return type: void

Description: API to Initialize Channel of GPIO as Left LED

1. ➢ void LL\_ON (u8 ChannelId)

Name: LL\_ON

Arguments:

* o Name: ChannelId
* o Type: u8
* o Range: 0:10
* o Description: Channel of GPIO connected to Left LED

Return type: void

Description: API to make Left LED ON

1. ➢ void LL\_OFF (void)

Name: LL\_OFF

Arguments:

* o Name: ChannelId
* o Type: u8
* o Range: 0:10
* o Description: Channel of GPIO connected to Left LED

Return type: void

Description: API to make Left LED OFF

**Buzzer Driver:**

1. ➢ void Buzzer\_Init (u8 ChannelId)

Name: Buzzer\_Init

Arguments:

* o Name: ChannelId
* o Type: u8
* o Range: 0:10
* o Description: Channel of GPIO connected to Buzzer

Return type: void

Description: API to initialize Channel of GPIO as Buzzer

Name: Buzzer\_ON

Arguments:

* o Name: ChannelId
* o Type: u8
* o Range: 0:10
* o Description: Channel of GPIO connected to Buzzer

Return type: void

Description: API to make Buzzer ON

1. ➢ void Buzzer\_OFF (u8 ChannelId)

Name: Buzzer\_OFF

Arguments:

* o Name: ChannelId
* o Type: u8
* o Range: 0:10
* o Description: Channel of GPIO connected to Buzzer

Return type: void

Description: API to make Buzzer ON

**CAN Handler:**

To enable total abstraction, a handler will be added as a point of contact between the CAN manager and the can Protocol.

**CAN Manager:**

1. o Name: CANManager\_Init
2. o Return type: void
3. o Description: API for initialization of communication
4. ➢ void CANManager\_Receive(void)
5. o Name: CANManager\_Receive
6. o Return type: void
7. o Description: API for Receiving messages (using CAN\_Read ())

**Output Manager:**

1. ➢ void OutputManager\_Init(void)

Name: OutputManager\_Init

Description: API for initialization of all Output devices by calling (LL\_Init(), RL\_Init(),Buzzer\_Init())

Name: OutputManager\_Control

Description: API for controlling of all Output devices by calling (LL\_ON (), RL\_ON (), RL\_OFF (), LL\_OF (), Buzzer\_ON (), Buzzer\_OFF ())

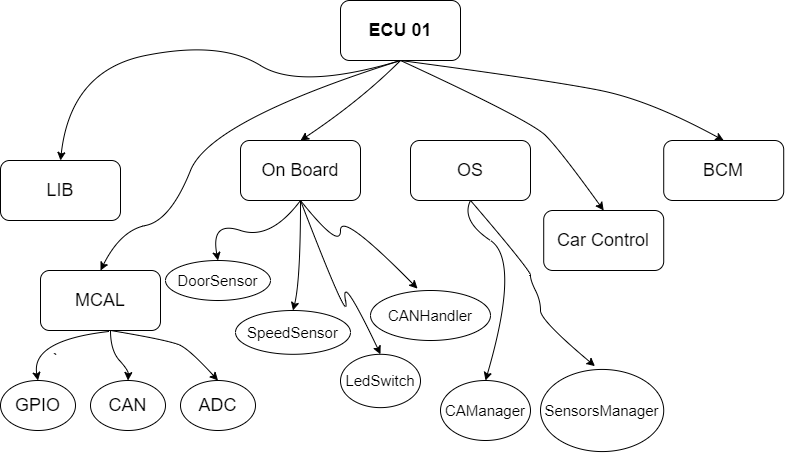
**Car Control:**

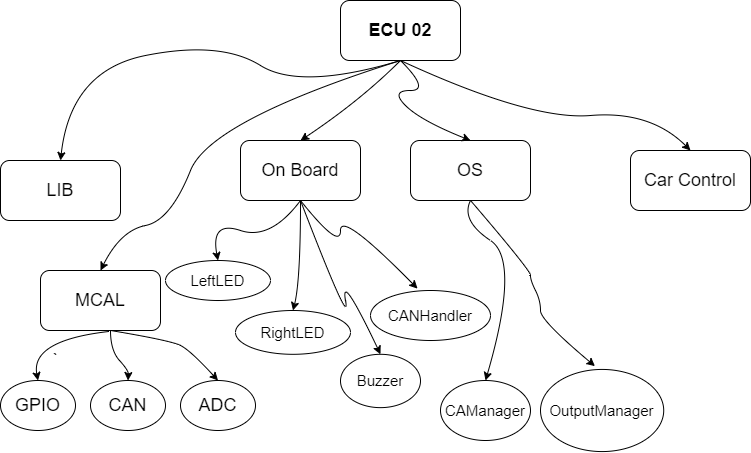
1. o Name: CommunicationManager\_Init
2. o Return type: void
3. o Description: this API Call the OS API to initialize Communication (CANManager\_Init ())
4. ➢ Void ReceivingMesseges\_Control(void)
5. o Name: ReceivingMesseges\_Control
6. o Return type: void

* o Description: this API Call OS API to start receiving can messages (CANManager\_Receive ())

1. ➢ void OutputDevices\_Init(void)
2. o Name: OutputDevices\_Init
3. o Return type: void
4. o Description: this API Call the OS API to initialize Output devices (OutputManager\_Init ())
5. ➢ void OutputDevices\_Control(void)
6. o Name: OutputDevices\_Control
7. o Return type: void
8. o Description: this API call the OS API to control output devices (using OutputManager\_Control())

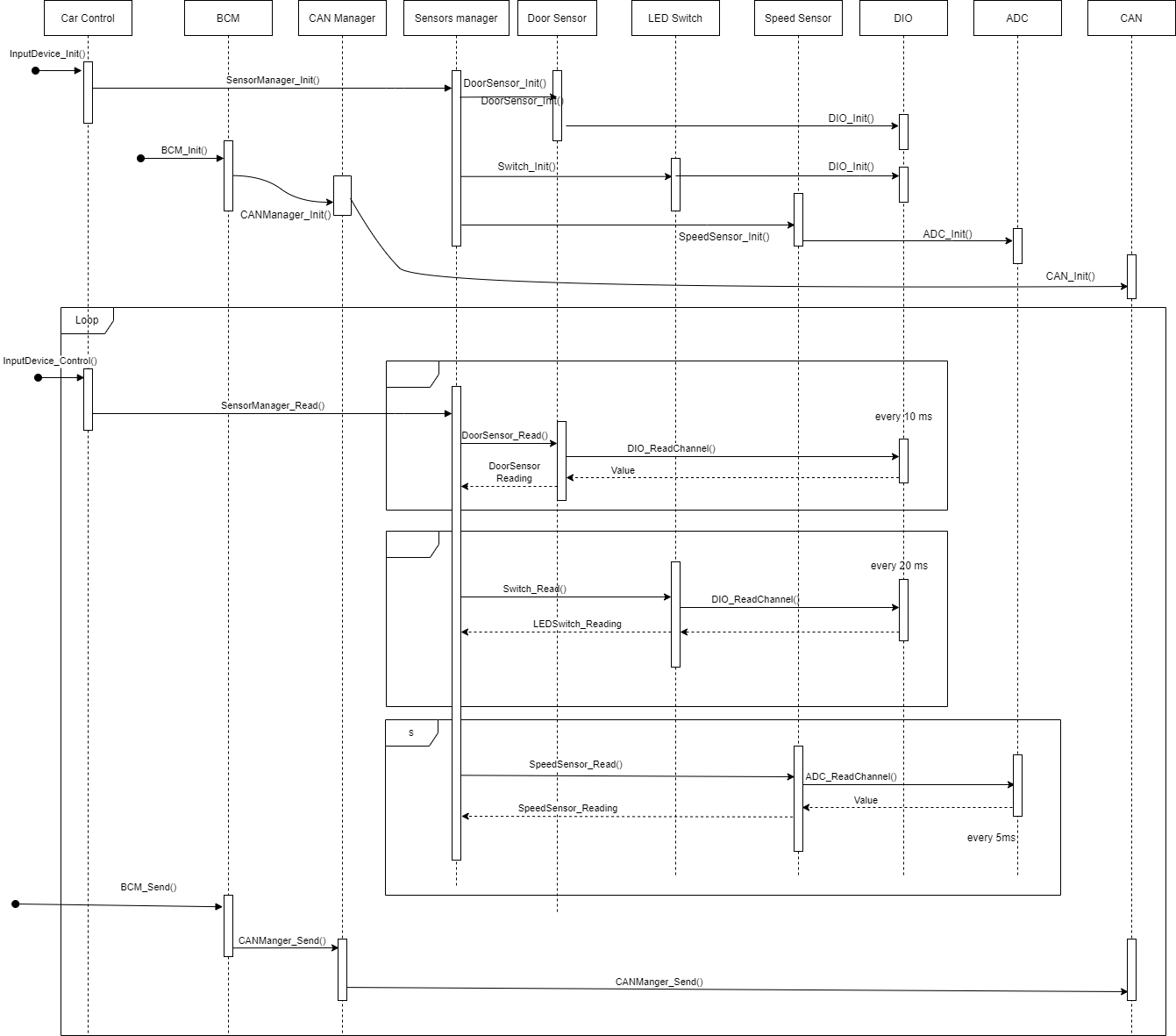
2: Folder Structure





3 Dynamic Design

ECU 1 Sequence diagram

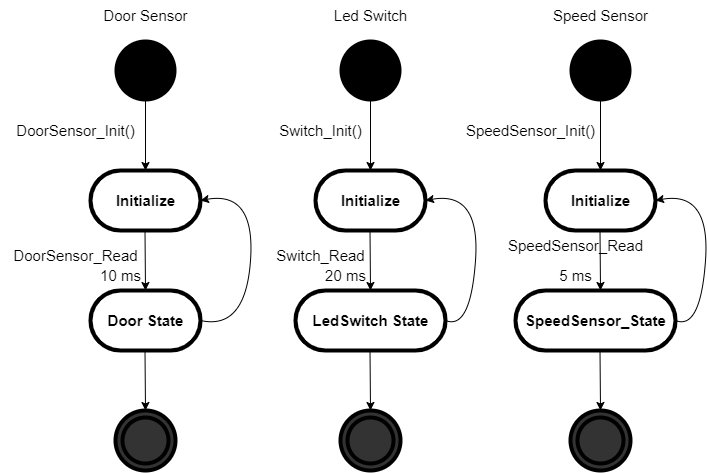


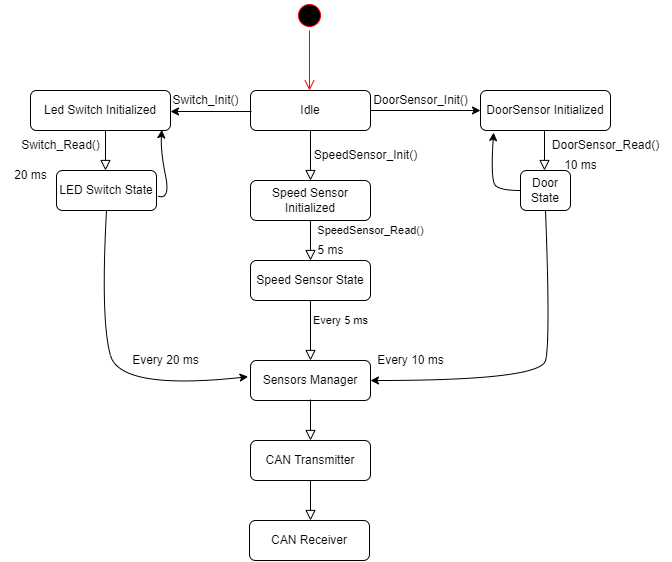
CPU Load

CPU Utilization = 100 - idle time = 100 - 65

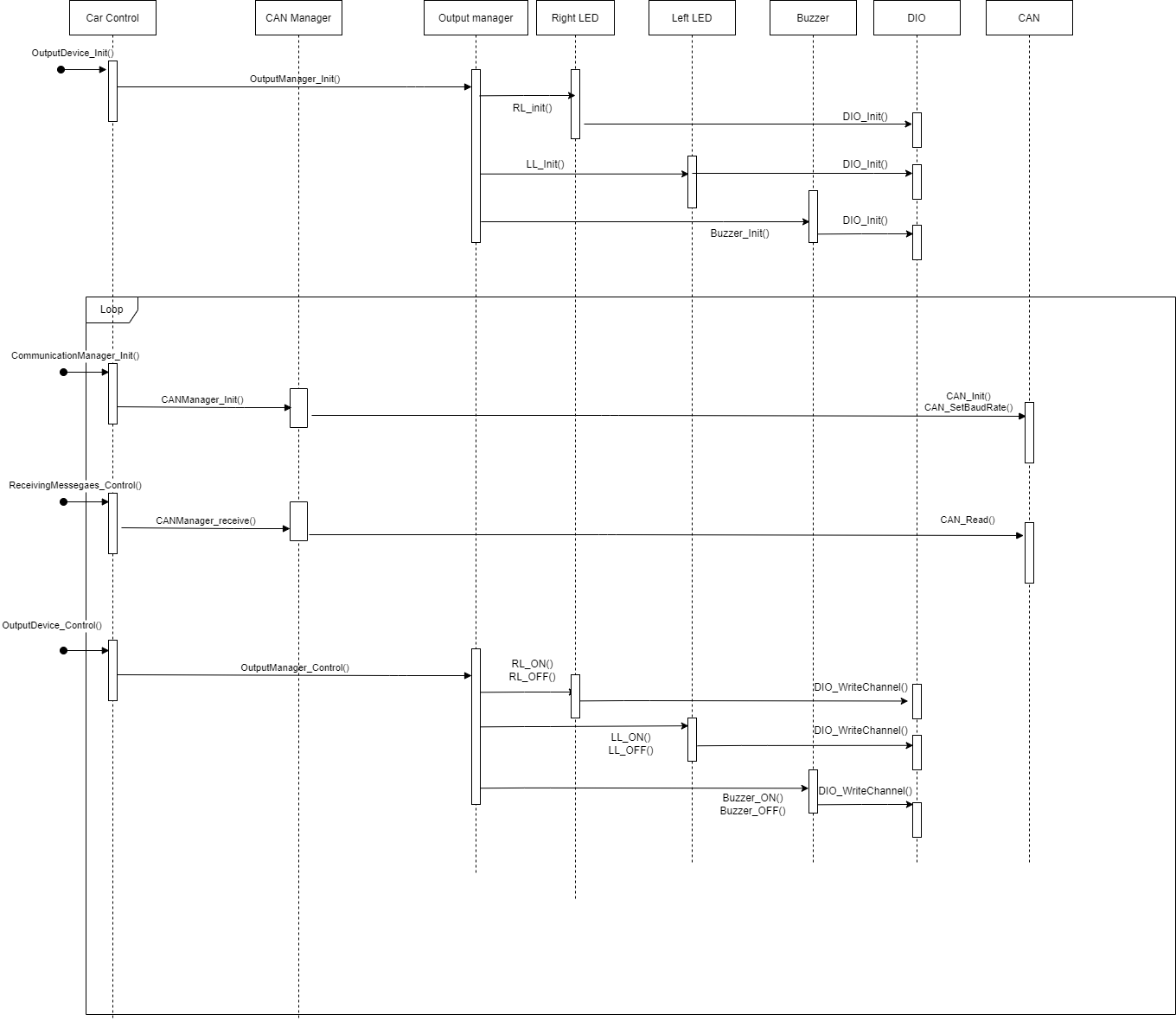
CPU Utilization = 35%

State Machine





ECU 2: Sequence diagram

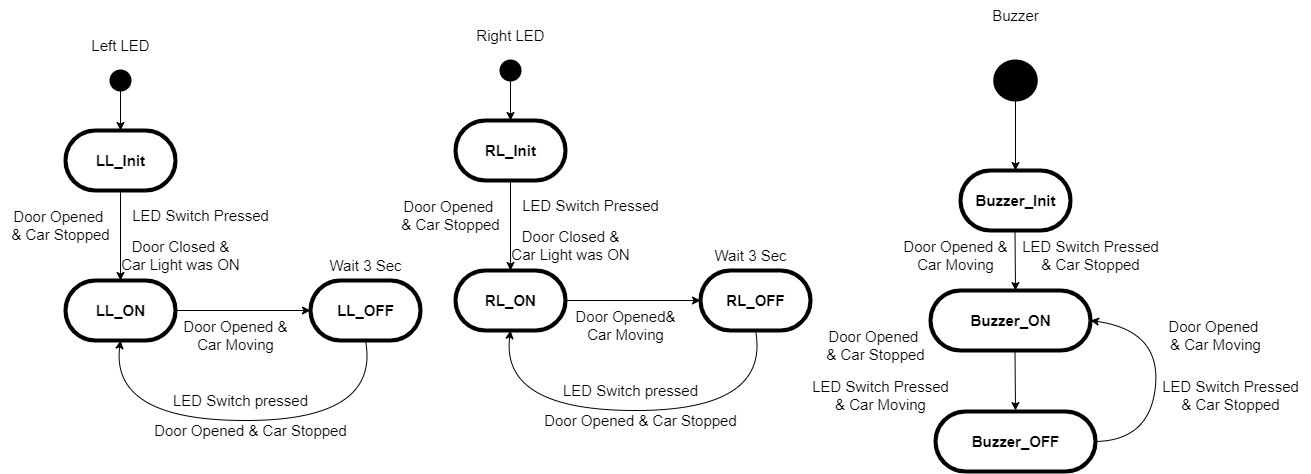


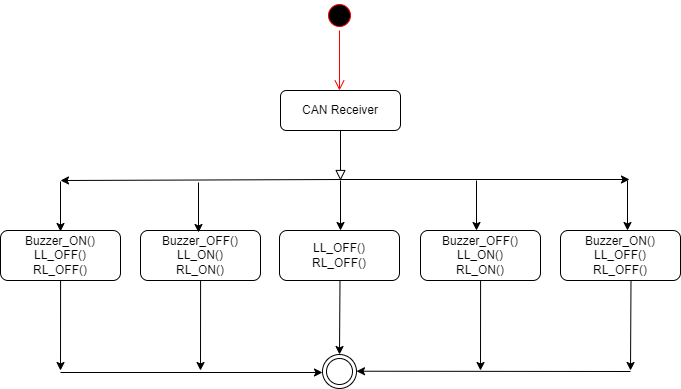
CPU Load

CPU Utilization = 100 - idle time = 100 - 65

CPU Utilization = 35%

State Machine





**4. Bus load**

A CAN frame has approximately 125 bits

Assume that we are using 500 kBit/s bit rate:

bit time = 1 / bit rate = 1 / (500 \* 1000) s = 2 \* 10-6 s = 2 μs

This means 1 bit will take 2 μs to transfer on bus when using 500 kBit/s

So the approximate time to transfer 1 frame is (2 μs/bit \* 125 bit) = 250 μs

Three messages are:

* • Door sensor message = 1 frame every 10 ms
* • Light switch message = 1 frame every 20 ms
* • Speed sensor message = 1 frame every 5 ms

1 frame every 10 ms = 100 frames every 1000 ms 1 frame every 20 ms = 50 frames every 1000 ms 1 frame every 5 ms = 200 frame every 1000 ms

Total frames = 350 frames every 1000 ms

Total time on bus = 350 \* 250 μs

Total time = 1000 ms

Bus load = ((350 \* 250) / (1000 \* 1000)) \* 100 % = 8.75 %