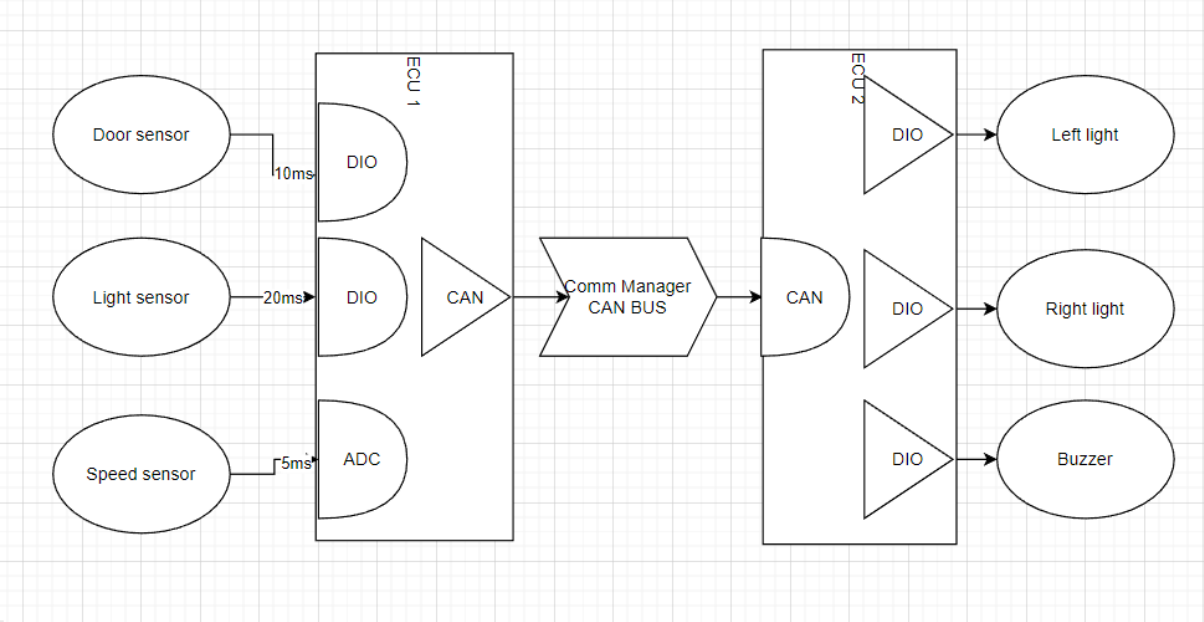
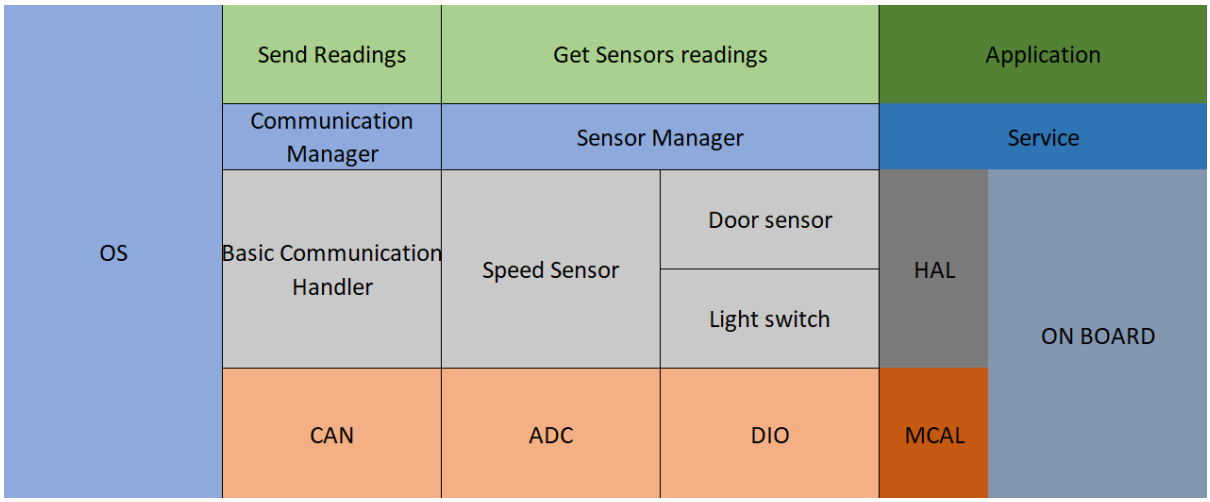
Block Diagram



Static design analysis

ECU 1

Layered Architecture



Components

- Speed Sensor
- Door Sensor
- Light Switch
- CAN
- ADC
- DIO

Modules

- Communication manager
- Communication handler
- Sensor manager
- Send readings
- Get sensor readings

APIs and TypeDefs

DIO

```c
/*============== Includes ==============*/
#include "../Common/TypeDefs.h"
/*============== TYPEDEFS ==============*/
/*Ports*/
typedef enum{
    PORTA,
    PORTB
}PORT_t;
/*Pins*/
typedef enum{
    PIN0,
    PIN1
}PIN_t;
/*Pin Modes*/
typedef enum{
    INPUT,
    OUTPUT
}MODE_t;
/*output levels*/
typedef enum{
    LOW,
    HIGH
}DATA_LEVEL_t;

/* Config structure */
typedef struct dioConfig{
    PORT_t port;
    PIN_t pin;
    MODE_t mode;
}st_DIO_Config;

/*====================== Function Prototypes ======================*/

void DIO_Init(st_DIO_Config *configstruct);
void DIO_write(st_DIO_Config *configstruct, DATA_LEVEL_t output);
uint8_t DIO_read(st_DIO_Config *configstruct);
```

DIO_Init

Return: void

Parameters: st_DIO_Config .Contains Port , Pin , Mode of operation (see typedefs for details).

Function: initialized the assigned pin to output or input mode

---

DIO_write

Return: void

Parameters: st_DIO_Config .Contains Port and pin (see typedefs for details).

DATA_LEVEL_t HIGH or LOW

Function: outputs certain data to an assigned pin the config struct.

DIO_read

Return: uint8_t

Parameters: st_DIO_Config .Contains Port , Pin (see typedefs for details).

Function: read gpio level on an assigned pin in the config structure

CAN

```c
/*================ Includes ================*/
#include "../Common/TypeDefs.h"

/*================ TYPEDEFS ================*/

/*messages IDs*/
typedef enum{
    DoorSensorMsg,
    LightSwitchMsg,
    SpeedSensorMsg
}MSG_t;
typedef enum{
    CAN_125KBPS,
    CAN_500KBPS,
}SPEED_t;


/* Config structure */
typedef struct dioConfig{
    MSG_t messageID;
    uint8_t dataBytes:4;
    SPEED_t speed;

}st_CAN_Config;

/*====================== Function Prototypes ======================*/

void CAN_Init(st_CAN_Config *configstruct);
void CAN_write(st_CAN_Config *configstruct, uint8_t[] data);
uint8_t* CAN_read(st_CAN_Config *configstruct);
```

CAN_Init

Return: void

Parameters: st_ CAN _Config .Contains message ID , number of data bytes , Speed of CAN (see typedefs for details).

Function: initialized the CAN configurations

CAN _write

Return: void

Parameters: : st_ CAN _Config .Contains message ID , number of data bytes , Speed of CAN (see typedefs for details).

Uint8_t arr[] contains the data to be sent through the bus

Function: Sends data bytes through the can bus

CAN _read

Return: uint8_t arr[]

Return: void

Parameters: : st_ CAN _Config .Contains message ID , number of data bytes , Speed of CAN (see typedefs for details).

Function: Receive incoming messages by the message ID

ADC

```c
/*=============== Includes ===============*/
#include "../Common/TypeDefs.h"

/*=============== TYPEDEFS ===============*/

/*messages IDs*/
typedef enum{
    ADC_CH0,
    ADC_CH1
}ADC_CH_t;

typedef enum{
    ADC_PRE8,
    ADC_PRE16
}ADC_Prescalars_t;

typedef enum{
    ADC_OneShot,
    ADC_FreeRunning
}ADC_Mode_t;


/* Config structure */
typedef struct adcConfig{
    ADC_CH_t channel;
    ADC_Prescalars_t prescalar;
    ADC_Mode_t mode;
}st_ADC_Config;

/*====================== Function Prototypes ======================*/
void ADC_Init(st_ADC_Config *configstruct);
uint16_t ADC_readChannel(st_ADC_Config *configstruct);
```

ADC_Init

Return: void

Parameters: st_ADC_Config contains the config parameters (channel,prescalar,mode) (see typedefs.)

Function: initializes the ADC configuration mode

ADC_readChannel

Return: uint16_t

Parameters: st_ADC_Config contains the config parameters (channel) (see typedefs.)

Function: Reads the assigned ADC channel and returns the value of ADC

HAL

Switch

```c
/*================ Includes ================*/
#include "../MCAL/DIO.h"

/*================ TYPEDEFS ================*/

typedef enum{
    CLOSED,
    OPENED
}LevelState_t;

/* Config structure */
typedef struct switchConfig{
    st_DIO_Config DIO;
}st_Switch_Config;

/*====================== Function Prototypes ======================*/
void Switch_Init(st_Switch_Config *configstruct);
LevelState_t Switch_read(st_Switch_Config *configstruct);
```

Switch_Init

Return: void

Parameters: st_Switch_Config holds dio port and pin and mode

Function: initialized switch as input at certain port and pin

---

Switch_read

Return: LevelState_t whether the switch is opened or closed

Parameters: st_Switch_Config holds dio port and pin and mode

Function: checks whether the switch is opened or closed

Speed

```c
/*=============== Includes ===============*/
#include "../MCAL/ADC.h"

/*=============== TYPEDEFS ===============*/



/* Config structure */
typedef struct speedConfig{
    st_ADC_Config ADC;
}st_Speed_Config;

/*====================== Function Prototypes ======================*/
void SpeedSensor_Init(st_Speed_Config *configstruct);
uint16_t SpeedSensor_Read(st_Speed_Config *configstruct);
```

SpeedSensor_Init

Return: void

Parameters: st_Speed_Config holds the adc init parameters

Function: initialized the adc of the used channel

SpeedSensor_Read

Return: uint16_t

Parameters: st_Speed_Config holds the adc channel

Function: Reads the ADC value of the assigned channel

DOOR

```c
/*=============== Includes ===============*/
#include "../MCAL/DIO.h"

/*=============== TYPEDEFS ===============*/

typedef enum{
    DoorClosed,
    DoorOpened
}DoorStatus_t;

/* Config structure */
typedef struct doorConfig{
    st_DIO_Config DIO;
}st_DoorSensor_Config;

/*====================== Function Prototypes ======================*/
void DoorSensor_Init(st_DoorSensor_Config *configstruct);
DoorStatus_t DoorSensor_read(st_DoorSensor_Config *configstruct);
```

DoorSensor_Init

Return: void

Parameters: st_DoorSensor_Config holds dio port and pin and mode

Function: initialized door sensor as input at certain port and pin

---

DoorSensor_read

Return: DoorStatus_t whether the door is opened or closed

Parameters: st_DoorSensor_Config holds dio port and pin and mode

Function: checks whether the door is opened or closed

Comm Handler

```c
/*================ Includes ================*/
#include "../MCAL/CAN.h"

/*================ Config ==================*/
#define CAN
/*================ TYPEDEFS ================*/


/* Config structure */
#ifdef CAN

    /*======================= Function Prototypes ======================
    void Data_SendViaCAN(st_CAN_Config *configstruct);
    uint8_t* Data_ReceiveViaCAN(st_CAN_Config *configstruct);

#endif
```

Data_SendViaCAN (interfaces with CAN module)

Return: void

Parameters: : st_ CAN _Config .Contains message ID , number of data bytes , Speed of CAN (see typedefs for details).

Uint8_t arr[] contains the data to be sent through the bus

Function: Sends data bytes through the can bus

Data_ReceiveViaCAN

Return: uint8_t arr[]

Parameters: : st_ CAN _Config .Contains message ID , number of data bytes , Speed of CAN (see typedefs for details).

Function: Receive incoming messages by the message ID

Service

Comm manager

```c
/*=============== Includes ===============*/
#include "../HAL/CommHandler.h"

/*=============== Config =================*/
#define USE_CAN
/*=============== TYPEDEFS ===============*/


/* Config structure */
#ifdef USE_CAN
    typedef struct COMMConfig{
        st_CAN_Config MSG;
    }st_COMM_Config;
#endif

/*====================== Function Prototypes ======================*/
void Data_Send(st_COMM_Config *configstruct,uint8_t[] data);
void Data_Receive(st_COMM_Config *configstruct);
```

Data_Send

Return: void

Parameters: st_COMM_Config chooses which communication config is used

Uint8_t[] holds the data to be sent

Function: Send data over a **predefined communication protocol**

Data_Receive

Return: Uint8_t[] holds the received data

Parameters: st_COMM_Config chooses which communication config is used

Function: Receiving data from **predefined comm protocol**

Sensor manager

```c
/*================ Includes ================*/
#include "../HAL/DoorSensor.h"
#include "../HAL/Switch.h"
#include "../HAL/SpeedSensor.h"


/*================ TYPEDEFS ================*/

typedef enum {
    DoorSensorID,
    Switch_ID,
    SpeedSensorID
}SensorIDs_t;

/*====================== Function Prototypes ======================*/
void            Sensor_init         (SensorIDs_t sensorID);
uint8_t         Sensor_readByte     (SensorIDs_t sensorID);
LevelState_t    Sensor_readLevel    (SensorIDs_t sensorID);
uint16_t        Sensor_read2Bytes   (SensorIDs_t sensorID);
```

Sensor_init

Return: void

Parameters: SensorIDs_t holds the sensor id to be dealt with

Function: initializes the sensor config and pin

Sensor_readByte

Return: uint8_t

Parameters: SensorIDs_t holds the sensor id to be dealt with

Function: Read the sensor output data as 8 bit data

Sensor_readLevel

Return:LevelState_t

Parameters: SensorIDs_t holds the sensor id to be dealt with

Function: Read the sensor output data as HIGH or LOW

Sensor_read2Bytes

Return:uint16_t

Parameters: SensorIDs_t holds the sensor id to be dealt with

Function: Read the sensor output data as 16 bit data

App

```c
/*=============== Includes ===============*/
#include "../Service/CommManager.h"
#include "../Service/SensorManager.h"


/*====================== Function Prototypes ====================*/


/* 3 Periodic functions pushing data into queue*/
LevelState_t GetDoorState(SensorIDs_t sensorID);              /*Called every 10ms by operating system*/
LevelState_t GetSwitchState(SensorIDs_t sensorID);            /*Called every 20ms by operating system*/
uint16_t     GetSpeed(SensorIDs_t sensorID);                  /*Called every 5ms by operating system*/


/*Event triggered function sends whenever queue is available*/
void SendDataToActuators(uint8_t data);                       /*Sending data via CAN bus*/
```

GetDoorState

Return: LevelState_t

Parameters: SensorIDs_t holds the sensor ID to hold the door sensor

Function: check for the door status

GetSwitchState

Return: LevelState_t

Parameters: SensorIDs_t holds the sensor ID to hold the switch sensor

Function: check for the switch status

GetSpeed

Return: uint16_t

Parameters: SensorIDs_t holds the sensor ID to hold the speed sensor

Function: check for the speed

SendDataToAcuators

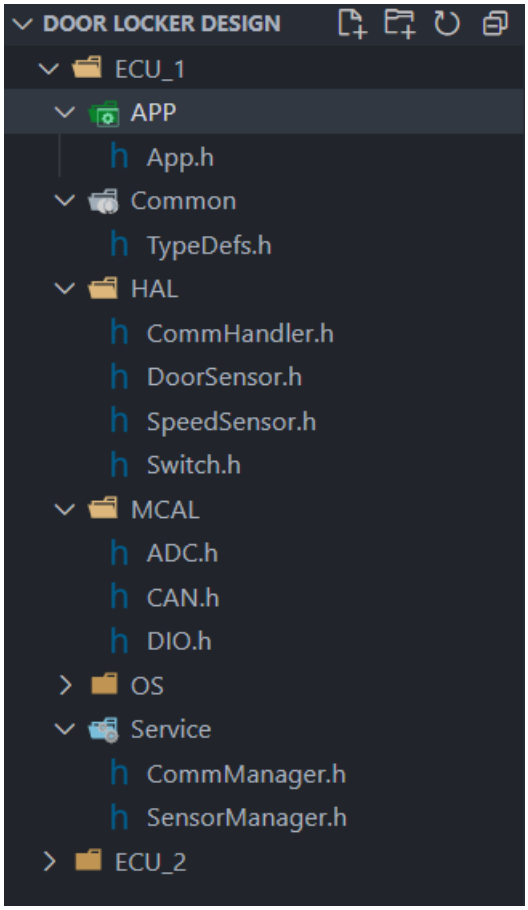
Return: void

Parameters: uint8_t

Function: sends the passed argument through the communication manager

---

Folder Structure



ECU 2

Buzzer

```c
/*================ Includes ================*/
#include "../MCAL/DIO.h"

/*================ TYPEDEFS ================*/



/* Config structure */
typedef struct buzzerConfig{
    st_DIO_Config DIO;
}st_Buzzer_Config;

/*====================== Function Prototypes ======================*/
void Buzzer_Init(st_Buzzer_Config *configstruct);
void Buzzer_ON(st_Buzzer_Config *configstruct);
void Buzzer_OFF(st_Buzzer_Config *configstruct);
void Buzzer_write(st_Buzzer_Config *configstruct, DATA_LEVEL_t output);
```

Buzzer_Init

Return: void

Parameters: st_Buzzer_Config holds dio port and pin (See TypeDefs)

Function: initializes the pin of the buzzer as output

Buzzer_ON

Return: void

Parameters: st_Buzzer_Config holds dio port and pin (See TypeDefs)

Function: writes high on the buzzer

Buzzer_OFF

Return: void

Parameters: st_Buzzer_Config holds dio port and pin (See TypeDefs)

Function: writes low on the buzzer pin

Buzzer_write

Return: void

Parameters: st_Buzzer_Config holds dio port and pin (See TypeDefs)

DATA_LEVEL_t either high or low

Function: write the given output to the buzzer pin

Light

```c
/*================ Includes ================*/
#include "../MCAL/DIO.h"

/*================ TYPEDEFS ================*/


/* Config structure */
typedef struct lightConfig{
    st_DIO_Config DIO;
}st_Light_Config;

/*====================== Function Prototypes ======================*/
void Light_Init(st_Light_Config *configstruct);
void Light_ON(st_Light_Config *configstruct);
void Light_OFF(st_Light_Config *configstruct);
void Light_write(st_Light_Config *configstruct,DATA_LEVEL_t out);
```

Light_Init

Return: void

Parameters: st_Light_Config holds dio port and pin (See TypeDefs)

Function: initializes the pin of the Light as output

Light _ON

Return: void

Parameters: st_Light_Config holds dio port and pin (See TypeDefs)

Function: writes high on the Light pin

Light _OFF

Return: void

Parameters: st_Light_Config holds dio port and pin (See TypeDefs)

Function: writes low on the Light pin

Light _write

Return: void

Parameters: st_Light_Config holds dio port and pin (See TypeDefs)

DATA_LEVEL_t either high or low

Function: write the given output to the Light pin

Output manager

```c
/*=============== Includes ===============*/
#include "../HAL/Buzzer.h"
#include "../HAL/Light.h"


/*=============== TYPEDEFS ===============*/

typedef enum {
    LeftLightID,
    RightLight_ID,
    BuzzerID
}ActuatorIDs_t;

/*======================= Function Prototypes =======================*/
void         Output_init    (ActuatorIDs_t ActuatorID);
void         Ouput_ON       (ActuatorIDs_t ActuatorID);
void         Ouput_OFF      (ActuatorIDs_t ActuatorID);
```

Output_init

Return: void

Parameters: AcuatorIDs_t holds the actuator ID to be dealt with

Function: Initializes the given Actuator

Output_ON

Return: void

Parameters: AcuatorIDs_t holds the actuator ID to be dealt with

Function: Turn on the actuator passed

Output_OFF

Return: void

Parameters: AcuatorIDs_t holds the actuator ID to be dealt with

Function: Turn off the actuator passed

App

```c
/*=============== Includes ===============*/
#include "../Service/CommManager.h"
#include "../Service/OutputManager.h"


/*=============== TypeDefs ================*/
typedef enum{
    STATE_A,
    STATE_B,
    STATE_C
}STATES_t;

/*====================== Function Prototypes ====================*/


/*each API of these are called in OS Task*/

/*Pushes incoming data to queue*/
uint8_t* GetDataviaCAN(st_COMM_Config *configstruct);

/*Processes incoming data to select the state*/
STATES_t UpdateStateMachine();

/*Execute current state*/
void ExecuteState(STATES_t currentState);
```

GetDataViaCAN

Return:uint8_t

Parameters: st_COMM_Config (see commManager.h)

Function: receives data sent to the bus

UpdateStateMachine

Return: STATES_t holds current state

Parameters: void

Function: Processing incoming data to set the current state machine

---

ExecuteState

Return: void

Parameters: STATES_t

Function: Execute the current state machine that are predefined

---

Folder Structure