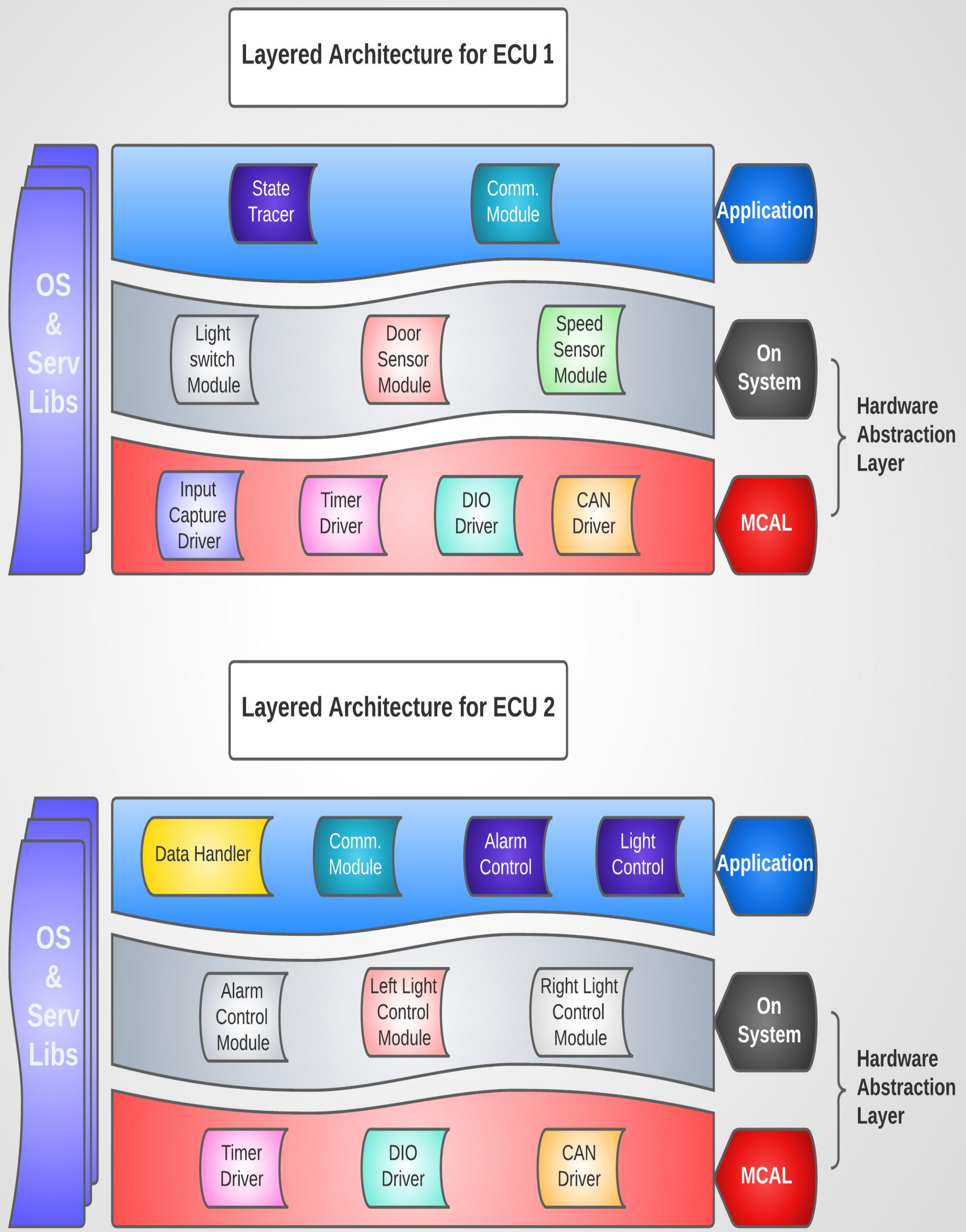
Automotive door control system

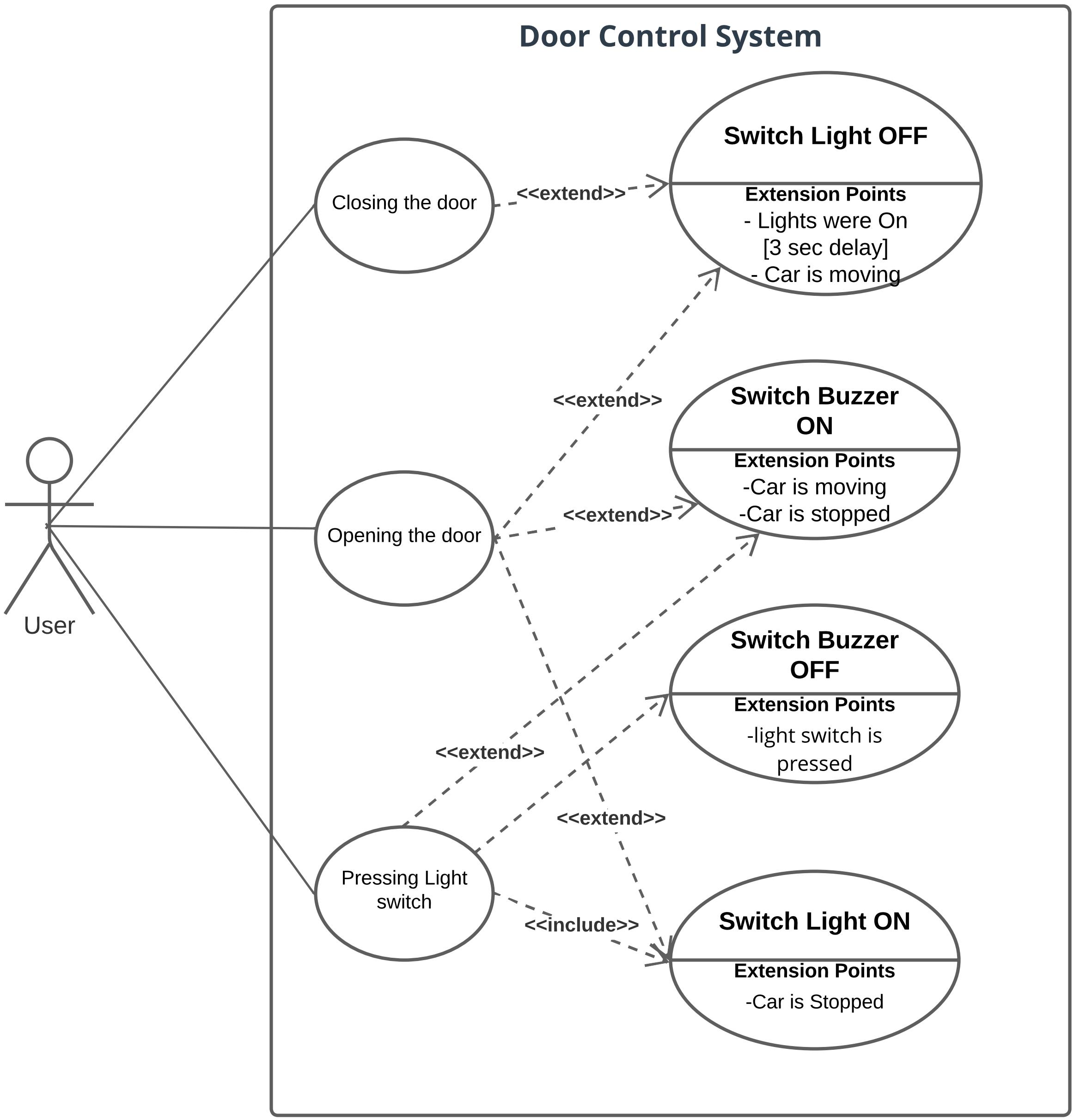
Static Design

In this paper we illustrate the static design for the automotive door control system, that includes the layered architecture of the system and full detailed description for the the components and their APIs.

- **First the layered architecture of the ECUs :**



USE CASES DIAGRAM



Here we specify each component and module in the ECU abstraction layer as well as the Low Layers Drivers. We start with the low layers to build such infrastructure for the higher layers as ECUAL and Application.

■ Low Layers

- Microcontroller Abstraction Layer (MCAL):

1. Timer Module Driver

As specified in HRS this target micro controller is up to attach with number of sensors that will utilize the timer module for timing management and synchronization of the communicati-on bus as it periodically transmit the tracing data on a CAN bus.

This Driver must provide APIs that utilize any of the hardware timers inside the MCU and generate accurate time based event triggering for specified number of times, API for providing the current counter of ticks as well as initialization functions.

API Type used for initialization the channels : _

```
/** @defgroup TIMER_ChannelType define
 * @brief TIMERs Channels define (the user uses this defines as insatnces for the config type )
 * @{
 */
#defien TIM_TypeDef uint32_t

#define TIM0 ((uint32_t)0X0000001)
#define TIM1 ((uint32_t)0X0000010)
#define TIM2 ((uint32_t)0X0000100)
#define TIM3 ((uint32_t)0X0001000)
#define TIM4 ((uint32_t)0X0010000)
#define TIM5 ((uint32_t)0X0100000)

/**
 * @}
 */
```

API Type used for configure the modes: _

```
/**
 * @brief Timer Channel Running Mode
 * @{
 */
typedef enum
{
    GPT_MODE_CONT,
    GPT_MODE_ONESHOT,
    GPT_MODE_SPEC

} GPT_ModeType;
/**
 * @}
 */
```

API Type used for struct the configuration parameters: _

```c
/**
 * @brief Timer ChannelS configuration type -> used in the initialization API
 */

typedef struct
{

    TIM_TypeDef Channel; /*!< TIM Channels def can be used                           */

    uint32_t TickFrequency; /*!< TIMER Frequency                                     */

    uint32_t TickValueMax; /*!< MAX Ticks for Channel                                */

    GPT_ModeType ChannelMode; /*!<Running mode [Continues, One-Shot or specified]  */

    void (*PeriodElapsedCallback)(void); /*!< TIM Period Elapsed Callback             */

} GPT_ConfgTypeDef;
/**
 * @}
 */
```

API functions used for initialization the driver and control operations : _

```c
/***************** Initialization functions ***********************/

void GPT_Init(GPT_ConfgTypeDef *ConfigPtr);

/*************** Timer Control functions *********************/

void GPT_StartTimer(TIM_TypeDef *Channel, uint32_t TicksValue);

void GPT_StopTimer(TIM_TypeDef *Channel);

uint32_t GPT_GetTimeElapsed(TIM_TypeDef *Channel);

uint32_t GPT_GetTimeRemaining(TIM_TypeDef *Channel);
```

2. DIO Module Driver

Digital input/output will be used by the ECUAL layer to communication with sensor and switches attached to the MCU.

It’s required to Provide full functional APIs to control the operation of the GPIO Module from reading and writing data and also controlling external interrupts on the pins.

API Type used for initialization the Pins :_

```c
/*- CONSTANTS -----------------------------------------------*/
/** @defgroup GPIO_pins_define GPIO pins define
 * @{
 */
#define PIN_TypeDef    uint16_t

#define GPIO_PIN_0 ((uint16_t)0x0001) /* Pin 0 selected    */
#define GPIO_PIN_1 ((uint16_t)0x0002) /* Pin 1 selected    */
#define GPIO_PIN_2 ((uint16_t)0x0004) /* Pin 2 selected    */
#define GPIO_PIN_3 ((uint16_t)0x0008) /* Pin 3 selected    */
#define GPIO_PIN_4 ((uint16_t)0x0010) /* Pin 4 selected    */
#define GPIO_PIN_5 ((uint16_t)0x0020) /* Pin 5 selected    */
#define GPIO_PIN_6 ((uint16_t)0x0040) /* Pin 6 selected    */
#define GPIO_PIN_7 ((uint16_t)0x0080) /* Pin 7 selected    */
/**
 * @}
 */
```

API Type used for specify the required PORT to control:_

```c
/** @defgroup GPIO_pins_define GPIO Ports define
 * @{
 */
#define PORT_TypeDef uint16_t

#define GPIO_PORTA ((PORT_TypeDef)0x0001) /* PORT A selected    */
#define GPIO_PORTB ((PORT_TypeDef)0x0002) /* PORT B selected    */
#define GPIO_PORTC ((PORT_TypeDef)0x0004) /* PORT C selected    */
#define GPIO_PORTD ((PORT_TypeDef)0x0008) /* PORT D selected    */
#define GPIO_PORTE ((PORT_TypeDef)0x0010) /* PORT E selected    */
#define GPIO_PORTF ((PORT_TypeDef)0x0020) /* PORT F selected    */

/**
 * @}
 */
```

API Type enum used to read and control the pins state:_

```c
/* @brief  GPIO Bit SET and Bit RESET enumeration
 */
typedef enum
{
    PIN_RESET = 0,
    PIN_SET

} GPIO_PinState;
```

API Types used for the configuration parameters:_

```
/** @defgroup GPIO_TYPES for using in initialization
 * @{
 */
#define MODE_TypeDef    uint32_t
#define PULL_TypeDef    uint32_t
#define ALTER_TypeDef   uint32_t
/**
 * @}
 */
```

API Types used to configure the operation modes of the pins:_

Note : it's up for the developer to define the required macros
for the bits positions according to the target used.

```
/** @defgroup GPIO_Private_Constants GPIO Private Constants
 * @{
 */
#define GPIO_MODE_Pos 0U
#define GPIO_MODE (0x3UL << GPIO_MODE_Pos)
#define MODE_INPUT (0x0UL << GPIO_MODE_Pos)
#define MODE_OUTPUT (0x1UL << GPIO_MODE_Pos)
#define MODE_AF (0x2UL << GPIO_MODE_Pos)
#define MODE_ANALOG (0x3UL << GPIO_MODE_Pos)

#define OUTPUT_TYPE_Pos 4U
#define OUTPUT_TYPE (0x1UL << OUTPUT_TYPE_Pos)
#define OUTPUT_PP (0x0UL << OUTPUT_TYPE_Pos)
#define OUTPUT_OD (0x1UL << OUTPUT_TYPE_Pos)

#define EXTI_MODE_Pos 16U
#define EXTI_MODE (0x3UL << EXTI_MODE_Pos)
#define EXTI_IT (0x1UL << EXTI_MODE_Pos)

#define TRIGGER_MODE_Pos 20U
#define TRIGGER_MODE (0x7UL << TRIGGER_MODE_Pos)
#define TRIGGER_RISING (0x1UL << TRIGGER_MODE_Pos)
#define TRIGGER_FALLING (0x2UL << TRIGGER_MODE_Pos)

/**
 * @}
 */
```

API Type used to struct the configuration parameters and passing to the initializing API:_

```c
/**
 * @brief GPIO Init structure definition
 */
typedef struct
{
    PORT_TypeDef Port; /*!< Specifies the GPIO Port to be configured.
                            This parameter can be any refrence of GPIOA_Type */

    PIN_TypeDef Pin; /*!< Specifies the GPIO pins to be configured.
                          This parameter can be any value of @ref GPIO_pins_define */

    MODE_TypeDef Mode; /*!< Specifies the operating mode for the selected pins.
                            This parameter can be a value of @ref GPIO_mode_define */

    PULL_TypeDef Pull; /*!< Specifies the Pull-up or Pull-Down activation for the selected pins.
                            This parameter can be a value of @ref GPIO_pull_define */

    ALTER_TypeDef Alternate; /*!< Peripheral to be connected to the selected pins.
                                  This parameter can be a value of @ref GPIO_Alternate_function_selection */
} GPIO_InitTypeDef;
```

API functions to initialize the DIO module and control operations:_

```c
/***************** Initialization function ***********************/
void GPIO_Init(GPIO_InitTypeDef *pstr_Init);

/*************** IO operation functions *********************/
GPIO_PinState GPIO_ReadPin(PORT_TypeDef *GPIOx, PIN_TypeDef GPIO_Pin);
void GPIO_WritePin(PORT_TypeDef GPIOx, PIN_TypeDef GPIO_Pin, GPIO_PinState PinState);
void GPIO_TogglePin(PORT_TypeDef GPIOx, PIN_TypeDef GPIO_Pin);

void GPIO_EXTI_CallbackSet(PORT_TypeDef GPIOx , , PIN_TypeDef GPIO_Pin , void (* EXTI_Callback )(void));

/**
 * @}
 */
```

3. Input Capture Module

Dealing with sensors requires signals measurements, that’s why the IC Driver must provide such APIs functions to measure the timing between rising and falling edges of the signal coming from the sensors, IC Driver utilize a timer unit on the selected target.

ECUAL layer depend on this driver to implement its components APIs. So that, it must be implemented accurately in order to evaluate such correct data from the sensors.

Here we illuminate the Module API types and functions ..

API type to specify the channels to be configured:_

```
/** @defgroup IC_Channel _Selection
 * @{
 */
#define IC_Channel_TypeDef uint32_t

#define IC_CHANNEL_1 ((IC_Channel_TypeDef)0x00000001) /*!< IC Channel 1 is selected */
#define IC_CHANNEL_2 ((IC_Channel_TypeDef)0x00000002) /*!< IC Channel 2 is selected */
#define IC_CHANNEL_3 ((IC_Channel_TypeDef)0x00000004) /*!< IC Channel 3 is selected */
#define IC_CHANNEL_4 ((IC_Channel_TypeDef)0x00000008) /*!< IC Channel 4 is selected */
/**
 * @}
 */
```

API type to select the utilized Timer to configure in IC mode :_

```
/** @defgroup TIM_Input_Capture_Selection TIM Input Capture Selection
 * @{
 */
#define TIM_SEL_TypeDef uint32_t
/*!< TIM1 Input 1, 2, 3 or 4 is selected to be connected to IC1, IC2, IC3 or IC4, respectively */
#define TIM1_ICSELECTION ((TIM_SEL_TypeDef)0x00000001)

/*!< TIM2 Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively */
#define TIM2_ICSELECTION ((TIM_SEL_TypeDef)0x00000002)

/*!< TIM3 Input 1, 2, 3 or 4 is selected to be connected to IC2, IC1, IC4 or IC3, respectively */
#define TIM2_ICSELECTION ((TIM_SEL_TypeDef)0x00000003)
/**
 * @}
 */
```

API type used to identify the Input Capture Polarity :_

```
/** @defgroup Input_Capture_Polarity Input Capture Polarity
 * @{
 */
#define ICPolarity_TypeDef uint32_t

/*!< Capture triggered by rising edge on timer input                */
#define ICPOLARITY_RISING ((ICPolarity_TypeDef)0x00000001)

/*!< Capture triggered by falling edge on timer input               */
#define ICPOLARITY_FALLING ((ICPolarity_TypeDef)0x00000002)

/*!< Capture triggered by both rising and falling edges on timer input*/
#define ICPOLARITY_BOTHEDGE ((ICPolarity_TypeDef)0x00000003)
/**
 * @}
 */
```

API type to struct the configuration parameters of the module and pass it to the initialization API function.:_

```c
/**
 * @brief  TIM Input Capture Configuration Structure definition
 */
typedef struct
{
    IC_Channel_TypeDef ICChannel; /*!< Specifies the channel of the input Capture.
                                       This parameter can be a value of @ref IC_Channel _Selection */

    ICPolarity_TypeDef ICPolarity; /*!< Specifies the active edge of the input signal.
                                         This parameter can be a value of @ref Input_Capture_Polarity */

    TIM_SEL_TypeDef ICSelection; /*!< Specifies the input.
                                       This parameter can be a value of @ref Input_Capture_Selection */

    uint32_t ICPrescaler; /*!< Specifies the Input Capture Prescaler.
                               This parameter can be a value of @ref Input_Capture_Prescaler */

} IC_InitTypeDef;
```

API functions to initialize the IC Module and control its operation.

```c
/***************** Initialization functions ***********************/

IC_StatusTypeDef IC_Init(IC_InitTypeDef *prsInit);
IC_StatusTypeDef IC_DeInit(IC_Channel_TypeDef ICChannel);

/*************** Timer Control functions *********************/
/* Blocking mode: Polling */
HAL_StatusTypeDef HAL_TIM_IC_Start(IC_Channel_TypeDef Channel);
HAL_StatusTypeDef HAL_TIM_IC_Stop(IC_Channel_TypeDef Channel);
/* Non-Blocking mode: Interrupt */
HAL_StatusTypeDef HAL_TIM_IC_Start_IT(IC_Channel_TypeDef Channel);
HAL_StatusTypeDef HAL_TIM_IC_Stop_IT(IC_Channel_TypeDef Channel);

IC_StatusTypeDef HAL_TIM_IC_GetState(IC_Channel_TypeDef Channel);

void IC_CallbackSet(IC_Channel_TypeDef Channel, void (* IC_Callback)(void));
```

***

4. CAN Module Driver

It’s specified that the first ECU target will pridoically transmit the collected data from the sensors on a CAN bus. In the higher levels we declared the communication module that will need the CAN driver APIs to achieve its purposes. So we define drivers APIs that will facilitate the required communication specifications.

API type for the CAN Module status :_

```
/* Exported types ------------------------------------------------------------*/
/** @defgroup CAN_Exported_Types CAN Exported Types
  * @{
  */
/**
  * @brief  CAN State structures definition
  */
typedef enum
{
    CAN_STATE_RESET = 0x00U,         /*!< CAN not yet initialized or disabled */
    CAN_STATE_READY = 0x01U,         /*!< CAN initialized and ready for use   */
    CAN_STATE_LISTENING = 0x02U,     /*!< CAN receive process is ongoing      */
    CAN_STATE_SLEEP_PENDING = 0x03U, /*!< CAN sleep request is pending        */
    CAN_STATE_SLEEP_ACTIVE = 0x04U,  /*!< CAN sleep mode is active            */
    CAN_STATE_ERROR = 0x05U          /*!< CAN error state                     */

} CAN_StateTypeDef;
```

API type to determine the operation mode of the module on the bus:

```
/** @defgroup CAN_operating_mode CAN Operating Mode
  * @{
  */
#define CAN_MODE_NORMAL (0x00000000U)                          /*!< Normal mode   */
#define CAN_MODE_LOOPBACK ((uint32_t)CAN_BTR_LBKM)             /*!< Loopback mode */
#define CAN_MODE_SILENT ((uint32_t)CAN_BTR_SILM)               /*!< Silent mode   */
/**
  * @}
  */
```

API type for definition of the Transmit header structure :_

```
/**
  * @brief  CAN Tx message header structure definition
  */
typedef struct
{
   uint32_t StdId; /*!< Specifies the standard identifier. */

   uint32_t IDE; /*!< Specifies the type of identifier for the message that will be transmitted. */

   uint32_t RTR; /*!< Specifies the type of frame for the message that will be transmitted.*/

   uint32_t DLC; /*!< Specifies the length of the frame that will be transmitted. */

} CAN_TxHeaderTypeDef;
```

API type to struct the initialization parameters to configure the module before starting operation :_

```c
/**
  * @brief  CAN init structure definition
  */
typedef struct
{
  uint32_t Prescaler;                  /*!< Specifies the length of a time quantum.*/

  uint32_t Mode;                       /*!< Specifies the CAN operating mode.
                                            This parameter can be a value of @ref CAN_operating_mode */

  uint32_t TimeSeg;                   /*!< Specifies the number of time quanta in Bit Segment.*/

  FunctionalState TimeTriggeredMode;   /*!< Enable or disable the time triggered communication mode.
                                            This parameter can be set to ENABLE or DISABLE. */

  FunctionalState AutoRetransmission;  /*!< Enable or disable the non-automatic retransmission mode.
                                            This parameter can be set to ENABLE or DISABLE. */

  FunctionalState TransmitFifoPriority;/*!< Enable or disable the transmit FIFO priority.
                                            This parameter can be set to ENABLE or DISABLE. */
} CAN_InitTypeDef;
```

API functions to allow initialization and control operations and manage interrupts for the Module:_

```c
/***************** Initialization function **********************/

CAN_StatusTypeDef HAL_CAN_Init(CAN_InitTypeDef *hcan);
CAN_StatusTypeDef HAL_CAN_DeInit(CAN_InitTypeDef *hcan);

/*************** control operation functions ********************/

CAN_StatusTypeDef CAN_Start(CAN_InitTypeDef *hcan);
CAN_StatusTypeDef CAN_Stop(CAN_InitTypeDef *hcan);
CAN_StatusTypeDef CAN_RequestSleep(CAN_InitTypeDef *hcan);
CAN_StatusTypeDef CAN_WakeUp(CAN_InitTypeDef *hcan);

CAN_StatusTypeDef CAN_AddTxMessage(CAN_InitTypeDef *hcan, CAN_TxHeaderTypeDef *pHeader,
                                                         uint8_t aData[], uint32_t *pTxMailbox);
CAN_StatusTypeDef CAN_AbortTxRequest(CAN_InitTypeDef *hcan, uint32_t TxMailboxes);

/* Interrupts management *****************************************************/
CAN_StatusTypeDef HAL_CAN_ActivateNotification(CAN_InitTypeDef *hcan, uint32_t ActiveITs);
CAN_StatusTypeDef HAL_CAN_DeactivateNotification(CAN_InitTypeDef *hcan, uint32_t InactiveITs);

/************** Callbacks functions ***************************************/
```

***

- ECU Abstraction Layer

Here we define the APIs of the ECUAL layer which on the application layer will depend directly. Where the components of this layer are different from the the two ECUs in the system we define each components group alone. Starting with the component of the ECU 1.

- ECU 1 Components :

1. Speed sensor

That sensor is supposed to provide API functions to get measurements from the hardware sensor attached to the target. Also to specify the car movement state for those situations we just care about the general state not a specific measure for the speed.

API type for retrieving the car movement state:_

```
/**
 * @brief  Movement State structures definition
 */
typedef enum
{
    Moving_State = 0,     /* Car is moving  */
    Stopping_State = 1  /*  Car is stopped*/

} Movement_StateTypeDef;
```

API function to control operation and retrieve measurements ans state:_

```
/***************** Initialization function ***********************/

void SpeedSens_init(void);

/*************** control operation functions *********************/

SpeedMeasure SpeedSens_getMeasure(void);
Movement_StateTypeDef SpeedSens_getMovState(void);
```

2. Door Sensor

That sensor is supposed to provide APIs functions to get doors state as well as Callbacks on changing in either state.

API type to identify the state of the car doors:_

```
/**
 * @brief  Door State structures definition
 */
typedef enum
{
    Door_Opened = 0, /* At least on the car's doors is opened   */
    Doors_Closed = 1 /*  All the doors are closed               */

} Door_StateTypeDef;
```

APIs functions to initialization and control operation and set Callbacks handlers:_

```
/***************** Initialization function ***********************/

void DoorSens_init(void);

/*************** control operation functions ********************/

Door_StateTypeDef DoorSens_getState(void);

void DoorSens_SetOpenCallback(void (*openState_CallBack)(void));
void DoorSens_SetCloseCallback(void (*closeState_CallBack)(void));
```

3. Light switch

This component will be responsible for reading the light switch input and get state of the switch on that moment it’s API called.

To make sure that the value returned from reading the switch is accurate and not effect by any kind on noise on the pins we assume the implementation will take care of that by reading the pins state by specified number of times on specified periods which could be controlled by the component APIs.

API type to identify the current state of the switch:_

```c
/**
 * @brief  Door State structures definition
 */
typedef enum
{

   SWITCH_OFF_STATE = 1, /*  LIGHT SWITCH IS NOT PRESSED    */
   SWITCH_ON_STATE = 0   /* LIGHT SWITCH IS PRESSED         */

} SWITCH_StateTypeDef;
```

API type used to define the filter numbers of samples used to decide the state of the switch and the period between each sample

```c
/**
 * @defgroup defines for the number of times to filter the reading values
 */
#define FilterNumType uint32_t
#define SamplePeriod uint32_t

#define FILTER_1_TIME ((FilterNumType)0X00000001)
#define FILTER_2_TIME ((FilterNumType)0X00000002)
#define FILTER_3_TIME ((FilterNumType)0X00000003)
#define FILTER_4_TIME ((FilterNumType)0X00000004)
#define FILTER_5_TIME ((FilterNumType)0X00000005)
```

API function to initialize the component and get switches state as well as set the callback functions that called on changing the state.

```c
/***************** Initialization function ***********************/

void LightSwitch_init(void);

/*************** control operation functions *********************/

SWITCH_StateTypeDef LightSW_getState(FilterNumType samples , SamplePeriod period);

void LightSW_SetON_Callback     (void (*SW_ON_CallBack) (void));
void LightSW_SetOFF_Callback    (void (*SW_OFF_CallBack)(void));
```

- ECU 2 Components :

Target Microcontroller in ECU 2 is supposed to attach with two light drivers for left and right lights and mosfet gate to control a buzzer or alarm.

we depend on the structure pattern to combine both the two light drivers controllers in one Module components and to provide API types to facilitate controlling both of them.

1. Light Control

Here we define types and api functions to control on both of the right and left lights.

API types that used in selecting between the two drivers to control:_

```
/**
 * @brief  Light Control structures definition
 */
typedef enum
{
    Left_Light_Select = 0,  /* Left Light driver is selected to control    */
    Right_Light_Select = 1, /* Right Light driver is selected to control   */
    Both_Light_Select = 2   /* Both Light driver is selected to control    */

} Light_SelectType;
```

API type used to provide status of the performed operation:_

```
/**
 * @brief  Light Control status definition
 */
typedef enum
{
    STATUS_OK = 0,     /* Contol operation has successfully done   */
    STATUS_NOT_OK = 1 /* Contol operation  has failed              */

} CTRL_StateTypeDef;
```

API functions that used to initialization and control operations:_

```
/***************** Initialization function ***********************/

void LightCtrl_init(void);

/*************** control operation functions ********************/

CTRL_StatusTypeDef LightCtrl_Set_ON (Light_SelectType lightSel);
CTRL_StatusTypeDef LightCtrl_Set_OFF (Light_SelectType lightSel);
```

2. Alarm Control

Here we define the APIs used to Control the Alarm operations.

API type used to identify the performed operation condition:_

```
/**
 * @brief  Alarm Control status definition
 */
typedef enum
{
    STATUS_OK = 0,     /* Contol operation has successfully done   */
    STATUS_NOT_OK = 1 /* Contol operation  has failed              */

} Alarm_StatusTypeDef;
```

API function to initialize the module and control operations:_

```
/***************** Initialization function ***********************/

void AlarmCtrl_init(void);

/*************** control operation functions ********************/

Alarm_StatusTypeDef AlarmCtrl_Set_ON(void);
Alarm_StatusTypeDef AlramCtrl_Set_OFF(void);
```