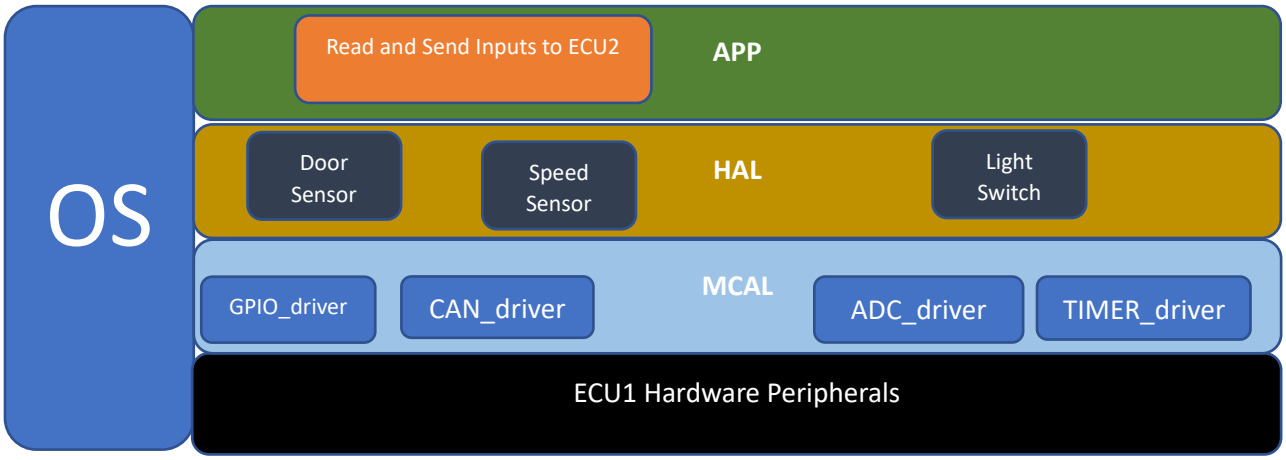
Static Design for ECU1

For ECU1:

- The Layered Architecture



- ECU Modules
  1- GPIO Module
  2- CAN Module
  3- Timer Module
  4- ADC Module

- Components attached to ECU1.
  1- Door Sensor
  2- Speed Sensor
  3- Light Switch

GPIO_driver.h APIs and Typedefs:

```c
/*******************MODES AND STATUS MACROS*****************************/
#define ENABLE          1U
#define DISABLE         0U
#define SET             ENABLE
#define RESET           DISABLE

/*
    @GPIO_PIN_NUMBER
    Pin numbers Macros
*/

#define GPIO_PIN_NO_0       0
#define GPIO_PIN_NO_1       1
#define GPIO_PIN_NO_2       2
#define GPIO_PIN_NO_3       3
#define GPIO_PIN_NO_4       4
#define GPIO_PIN_NO_5       5
#define GPIO_PIN_NO_6       6
#define GPIO_PIN_NO_7       7
#define GPIO_PIN_NO_8       8
#define GPIO_PIN_NO_9       9
#define GPIO_PIN_NO_10      10
#define GPIO_PIN_NO_11      11
#define GPIO_PIN_NO_12      12
#define GPIO_PIN_NO_13      13
#define GPIO_PIN_NO_14      14
#define GPIO_PIN_NO_15      15


/*
    @GPIO_PIN_MODE
    Pin Different Modes
*/

#define GPIO_MODE_IN            0
#define GPIO_MODE_OUT           1
#define GPIO_MODE_ALTFUN        2
#define GPIO_MODE_ANG           3
#define RISING_EDGE             4
#define FALLING_EDGE            5
#define RISING_FALLING          6

/*
```

```c
/*
    @GPIO_SPEED
    Pin different Speed
*/
#define LOW_SPEED                       0
#define MEDIUM_SPEED                    1
#define HIGH_SPEED                      2
#define VERY_HIGH_SPEED                 3


/*
    @GPIO_PUPD
    PULL-UP or PULL-DOWN resistor
*/

#define NO_PUPD         0
#define PULL_UP         1
#define PULL_DOWN       2


/*
    @GPIO_Output_Modes
    Output Modes
*/

#define PUSH_PULL   0
#define OPEN_DRAIN  1


/*
*   @GPIO_ALT
    Alternate Functionality
*/
#define AF0         0
#define AF1         1
#define AF2         2
#define AF3         3
#define AF4         4
#define AF5         5
#define AF6         6
#define AF7         7
#define AF8         8
#define AF9         9
#define AF10        10
#define AF11        11
#define AF12        12
#define AF13        13
#define AF14        14
```

```c
/******************GPIO pin configuration structure*************************/
typedef struct
{
    uint32_t pin_Num;               /*<refer to section @GPIO_PIN_NUMBER>*/
    uint32_t pin_Mode;              /*<refer to section @GPIO_PIN_MODE>*/
    uint32_t pin_Speed;             /*<refer to section @GPIO_SPEED>*/
    uint32_t pin_PUPD;              /*<refer to section @GPIO_PUPD>*/
    uint32_t pin_OutputMode;            /*<refer to section @GPIO_Output_Modes>*/
    uint32_t pin_AltFun;                /*<refer to section @GPIO_ALT>*/

}GPIO_PIN_CONFIG_t;

/***********************GPIO Handle structure*******************************/

typedef struct
{
    GPIO_TypeDef *pGPIOx;               //Pointer to the port x address
    GPIO_PIN_CONFIG_t gpio_pin_config;  //pin coniguration structure

}GPIO_handle_t;
/*******************************************************************************
*                   APIs supported by this driver
*******************************************************************************/

/*  1- GPIOx clock control */
void GPIO_CLK_CTRL(GPIO_TypeDef *pGPIOx, uint8_t status);


/*  2- GPIOx init and de-init */
void GPIO_Init(GPIO_handle_t *pGPIO_handle);
void GPIO_DeInit(GPIO_TypeDef *pGPIOx);

/*  3- Data Read and Write */
uint8_t GPIO_PinRead(GPIO_TypeDef *pGPIOx, uint8_t PinNumber);
uint16_t GPIO_PortRead(GPIO_TypeDef *pGPIOx);

void GPIO_PinWrite(GPIO_TypeDef *pGPIOx, uint8_t PinNumber, uint8_t data);
void GPIO_PinToggle(GPIO_TypeDef *pGPIOx, uint8_t PinNumber);
void GPIO_PortWrite(GPIO_TypeDef *pGPIOx, uint16_t data);
```

CAN_driver.h APIs and Typedefs:

```
** @defgroup CAN_Exported_Types CAN Exported Types
  * @{
  */
/**
  * @brief  HAL State structures definition
  */
typedef enum
{
  HAL_CAN_STATE_RESET             = 0x00U,  /*!< CAN not yet initialized or disabled */
  HAL_CAN_STATE_READY             = 0x01U,  /*!< CAN initialized and ready for use   */
  HAL_CAN_STATE_LISTENING         = 0x02U,  /*!< CAN receive process is ongoing      */
  HAL_CAN_STATE_SLEEP_PENDING     = 0x03U,  /*!< CAN sleep request is pending        */
  HAL_CAN_STATE_SLEEP_ACTIVE      = 0x04U,  /*!< CAN sleep mode is active            */
  HAL_CAN_STATE_ERROR             = 0x05U   /*!< CAN error state                     */

} HAL_CAN_StateTypeDef;

**
  * @}
  */

/**
  * @brief  CAN handle Structure definition
  */
typedef struct __CAN_HandleTypeDef
{
  CAN_TypeDef                 *Instance;                /*!< Register base address */

  CAN_InitTypeDef             Init;                     /*!< CAN required parameters */

  __IO HAL_CAN_StateTypeDef   State;                    /*!< CAN communication state */

  __IO uint32_t               ErrorCode;                /*!< CAN Error code.
 This parameter can be a value of @ref CAN_Error_Code */
}




/** @defgroup CAN_InitStatus CAN InitStatus
  * @{
  */
#define CAN_INITSTATUS_FAILED       (0x00000000U)  /*!< CAN initialization failed */
#define CAN_INITSTATUS_SUCCESS      (0x00000001U)  /*!< CAN initialization OK     */
/**
  * @}
  */
```

```c
/** @defgroup CAN_InitStatus CAN InitStatus
  * @{
  */
#define CAN_INITSTATUS_FAILED       (0x00000000U)  /*!< CAN initialization failed */
#define CAN_INITSTATUS_SUCCESS      (0x00000001U)  /*!< CAN initialization OK     */
/**
  * @}
  */

/** @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   */
#define CAN_MODE_SILENT_LOOPBACK    ((uint32_t)(CAN_BTR_LBKM | CAN_BTR_SILM)) /*!< Loopback combined with silent mode */

/* Initialization and de-initialization functions ****************************/
HAL_StatusTypeDef HAL_CAN_Init(CAN_HandleTypeDef *hcan);
HAL_StatusTypeDef HAL_CAN_DeInit(CAN_HandleTypeDef *hcan);

/* Control functions *********************************************************/
HAL_StatusTypeDef HAL_CAN_Start(CAN_HandleTypeDef *hcan);
HAL_StatusTypeDef HAL_CAN_Stop(CAN_HandleTypeDef *hcan);
HAL_StatusTypeDef HAL_CAN_AddTxMessage(CAN_HandleTypeDef *hcan, CAN_TxHeaderTypeDef *pHeader, uint8_t aData[], uint32_t *pTxMailbox);
HAL_StatusTypeDef HAL_CAN_GetRxMessage(CAN_HandleTypeDef *hcan, uint32_t RxFifo, CAN_RxHeaderTypeDef *pHeader, uint8_t aData[]);
```

Timer_driver.h APIs and Typedefs:

```c
/**
  * @brief  TIM Input Capture Configuration Structure definition
  */
typedef struct
{
  uint32_t  ICPolarity;  /*!< Specifies the active edge of the input signal.
                              This parameter can be a value of @ref TIM_Input_Capture_Polarity */

  uint32_t ICSelection;  /*!< Specifies the input.
                              This parameter can be a value of @ref TIM_Input_Capture_Selection */

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

  uint32_t ICFilter;     /*!< Specifies the input capture filter.
                              This parameter can be a number between Min_Data = 0x0 and Max_Data = 0xF */
} TIM_IC_InitTypeDef;

**
  * @brief  HAL State structures definition
  */
typedef enum
{
  HAL_TIM_STATE_RESET             = 0x00U,    /*!< Peripheral not yet initialized or disabled  */
  HAL_TIM_STATE_READY             = 0x01U,    /*!< Peripheral Initialized and ready for use    */
  HAL_TIM_STATE_BUSY              = 0x02U,    /*!< An internal process is ongoing              */
  HAL_TIM_STATE_TIMEOUT           = 0x03U,    /*!< Timeout state                               */
  HAL_TIM_STATE_ERROR             = 0x04U     /*!< Reception process is ongoing                */
} HAL_TIM_StateTypeDef;

/**
  * @brief  TIM Channel States definition
  */
typedef enum
{
  HAL_TIM_CHANNEL_STATE_RESET             = 0x00U,    /*!< TIM Channel initial state                         */
  HAL_TIM_CHANNEL_STATE_READY             = 0x01U,    /*!< TIM Channel ready for use                         */
  HAL_TIM_CHANNEL_STATE_BUSY              = 0x02U,    /*!< An internal process is ongoing on the TIM channel */
} HAL_TIM_ChannelStateTypeDef;
```

```c
/** @defgroup TIM_Exported_Types TIM Exported Types
  * @{
  */

/**
  * @brief  TIM Time base Configuration Structure definition
  */
typedef struct
{
  uint32_t Prescaler;         /*!< Specifies the prescaler value used to divide the TIM clock.
                                   This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF */

  uint32_t CounterMode;       /*!< Specifies the counter mode.
                                   This parameter can be a value of @ref TIM_Counter_Mode */

  uint32_t Period;            /*!< Specifies the period value to be loaded into the active
                                   Auto-Reload Register at the next update event.
                                   This parameter can be a number between Min_Data = 0x0000 and Max_Data = 0xFFFF.  */

  uint32_t ClockDivision;     /*!< Specifies the clock division.
                                   This parameter can be a value of @ref TIM_ClockDivision */

  uint32_t RepetitionCounter;  /*!< Specifies the repetition counter value. Each time the RCR downcounter
                                    reaches zero, an update event is generated and counting restarts
                                    from the RCR value (N).
                                    This means in PWM mode that (N+1) corresponds to:
                                       - the number of PWM periods in edge-aligned mode
                                       - the number of half PWM period in center-aligned mode
                                    GP timers: this parameter must be a number between Min_Data = 0x00 and
                                    Max_Data = 0xFF.
                                    Advanced timers: this parameter must be a number between Min_Data = 0x0000 and
                                    Max_Data = 0xFFFF. */

  uint32_t AutoReloadPreload;  /*!< Specifies the auto-reload preload.
                                   This parameter can be a value of @ref TIM_AutoReloadPreload */
} TIM_Base_InitTypeDef;

/**
  * @brief  TIM Input Capture Configuration Structure definition
  */
typedef struct
{
  uint32_t  ICPolarity;  /*!< Specifies the active edge of the input signal.
                              This parameter can be a value of @ref TIM_Input_Capture_Polarity */

  uint32_t ICSelection;  /*!< Specifies the input.
                              This parameter can be a value of @ref TIM_Input_Capture_Selection */

  uint32_t ICPrescaler;  /*!< Specifies the Input Capture Prescaler.
```

```c
/**
  * @brief  HAL Active channel structures definition
  */
typedef enum
{
  HAL_TIM_ACTIVE_CHANNEL_1        = 0x01U,    /*!< The active channel is 1     */
  HAL_TIM_ACTIVE_CHANNEL_2        = 0x02U,    /*!< The active channel is 2     */
  HAL_TIM_ACTIVE_CHANNEL_3        = 0x04U,    /*!< The active channel is 3     */
  HAL_TIM_ACTIVE_CHANNEL_4        = 0x08U,    /*!< The active channel is 4     */
  HAL_TIM_ACTIVE_CHANNEL_CLEARED  = 0x00U     /*!< All active channels cleared */
} HAL_TIM_ActiveChannel;


typedef struct
{
  TIM_TypeDef                 *Instance;         /*!< Register base address                          */
  TIM_Base_InitTypeDef        Init;             /*!< TIM Time Base required parameters              */
  HAL_TIM_ActiveChannel       Channel;          /*!< Active channel                                 */

  HAL_LockTypeDef             Lock;             /*!< Locking object                                 */
  __IO HAL_TIM_StateTypeDef   State;            /*!< TIM operation state                            */
  __IO HAL_TIM_ChannelStateTypeDef ChannelState[4];  /*!< TIM channel operation state               */
  __IO HAL_TIM_ChannelStateTypeDef ChannelNState[4]; /*!< TIM complementary channel operation state */
} TIM_HandleTypeDef;

/* Time Base functions ************************************************/
HAL_StatusTypeDef HAL_TIM_Base_Init(TIM_HandleTypeDef *htim);
HAL_StatusTypeDef HAL_TIM_Base_DeInit(TIM_HandleTypeDef *htim);
/* Blocking mode: Polling */
HAL_StatusTypeDef HAL_TIM_Base_Start(TIM_HandleTypeDef *htim);
HAL_StatusTypeDef HAL_TIM_Base_Stop(TIM_HandleTypeDef *htim);
/* Non-Blocking mode: Interrupt */
HAL_StatusTypeDef HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim);
HAL_StatusTypeDef HAL_TIM_Base_Stop_IT(TIM_HandleTypeDef *htim);
/* Timer Input Capture functions **************************************/
HAL_StatusTypeDef HAL_TIM_IC_Init(TIM_HandleTypeDef *htim);
HAL_StatusTypeDef HAL_TIM_IC_DeInit(TIM_HandleTypeDef *htim);

/* Blocking mode: Polling */
HAL_StatusTypeDef HAL_TIM_IC_Start(TIM_HandleTypeDef *htim, uint32_t Channel);
HAL_StatusTypeDef HAL_TIM_IC_Stop(TIM_HandleTypeDef *htim, uint32_t Channel);
/* Non-Blocking mode: Interrupt */
HAL_StatusTypeDef HAL_TIM_IC_Start_IT(TIM_HandleTypeDef *htim, uint32_t Channel);
HAL_StatusTypeDef HAL_TIM_IC_Stop_IT(TIM_HandleTypeDef *htim, uint32_t Channel);
```

ADC_driver.h APIs and Typedefs:

```c
typedef struct
{
  uint32_t ClockPrescaler;              /*!< Select ADC clock prescaler. The clock is common for
                                             all the ADCs.
                                             This parameter can be a value of @ref ADC_ClockPrescaler */
  uint32_t Resolution;                  /*!< Configures the ADC resolution.
                                             This parameter can be a value of @ref ADC_Resolution */
  uint32_t DataAlign;                   /*!< Specifies ADC data alignment to right (MSB on register bit 11 and LSB on register bit 0) (default setting)
                                             or to left (if regular group: MSB on register bit 15 and LSB on register bit 4, if injected group (MSB kept as signed value due to potential negative value after offset ap
                                             This parameter can be a value of @ref ADC_Data_align */
}ADC_InitTypeDef;

/**
  * @brief  Structure definition of ADC channel for regular group
  * @note   The setting of these parameters with function HAL_ADC_ConfigChannel() is conditioned to ADC state.
  *         ADC can be either disabled or enabled without conversion on going on regular group.
  */
typedef struct
{
  uint32_t Channel;               /*!< Specifies the channel to configure into ADC regular group.
                                       This parameter can be a value of @ref ADC_channels */
  uint32_t Rank;                  /*!< Specifies the rank in the regular group sequencer.
                                       This parameter must be a number between Min_Data = 1 and Max_Data = 16 */
  uint32_t SamplingTime;          /*!< Sampling time value to be set for the selected channel.
                                       Unit: ADC clock cycles
                                       Conversion time is the addition of sampling time and processing time (12 ADC clock cycles at ADC resolution 12 bits, 11 cycles at 10 bits, 9 cycles at 8 bits, 7 cycles at 6 bits
                                       This parameter can be a value of @ref ADC_sampling_times
                                       Caution: This parameter updates the parameter property of the channel, that can be used into regular and/or injected groups.
                                           If this same channel has been previously configured in the other group (regular/injected), it will be updated to last setting.
                                       Note: In case of usage of internal measurement channels (VrefInt/Vbat/TempSensor),
                                       sampling time constraints must be respected (sampling time can be adjusted in function of ADC clock frequency and sampling time setting)
                                       Refer to device datasheet for timings values, parameters TS_vrefint, TS_temp (values rough order: 4us min). */
  uint32_t Offset;                /*!< Reserved for future use, can be set to 0 */
}ADC_ChannelConfTypeDef;

/**
  * @brief  HAL ADC state machine: ADC states definition (bitfields)
  */
/* States of ADC global scope */
#define HAL_ADC_STATE_RESET             0x00000000U    /*!< ADC not yet initialized or disabled */
#define HAL_ADC_STATE_READY             0x00000001U    /*!< ADC peripheral ready for use */
#define HAL_ADC_STATE_BUSY_INTERNAL     0x00000002U    /*!< ADC is busy to internal process (initialization, calibration) */
#define HAL_ADC_STATE_TIMEOUT           0x00000004U    /*!< TimeOut occurrence */
```

```
                                    sampling time constraints must be respected (sampling time can be adjusted in function of ADC
                                    Refer to device datasheet for timings values, parameters TS_vrefint, TS_temp (values rough or
  uint32_t Offset;               /*!< Reserved for future use, can be set to 0 */
}ADC_ChannelConfTypeDef;

/**
  * @brief  HAL ADC state machine: ADC states definition (bitfields)
  */
/* States of ADC global scope */
#define HAL_ADC_STATE_RESET             0x00000000U    /*!< ADC not yet initialized or disabled */
#define HAL_ADC_STATE_READY             0x00000001U    /*!< ADC peripheral ready for use */
#define HAL_ADC_STATE_BUSY_INTERNAL     0x00000002U    /*!< ADC is busy to internal process (initialization, calibration) */
#define HAL_ADC_STATE_TIMEOUT           0x00000004U    /*!< TimeOut occurrence */



typedef struct

{
  ADC_TypeDef                 *Instance;                /*!< Register base address */

  ADC_InitTypeDef             Init;                     /*!< ADC required parameters */

  HAL_LockTypeDef             Lock;                     /*!< ADC locking object */

  __IO uint32_t               State;                    /*!< ADC communication state */

  __IO uint32_t               ErrorCode;                /*!< ADC Error code */
}ADC_HandleTypeDef;


/* Initialization/de-initialization functions **********************************/
HAL_StatusTypeDef HAL_ADC_Init(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_DeInit(ADC_HandleTypeDef *hadc);
/* I/O operation functions *****************************************************/
HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Stop(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_PollForConversion(ADC_HandleTypeDef* hadc, uint32_t Timeout);

HAL_StatusTypeDef HAL_ADC_PollForEvent(ADC_HandleTypeDef* hadc, uint32_t EventType, uint32_t Timeout);

HAL_StatusTypeDef HAL_ADC_Start_IT(ADC_HandleTypeDef* hadc);
HAL_StatusTypeDef HAL_ADC_Stop_IT(ADC_HandleTypeDef* hadc);
```

Door_sensor.h APIs and Typedefs:

```c
#ifndef DOOR_SENSOR_H
#define DOOR_SENSOR_H

/**
*@brief Door status
**/
typedef enum
{
    Door_closed,
    Doo_opened
}Door_status;

/**
*@brief Door Sensor APIs
**/

void Door_init(void);
void Door_Deinit(void);

Door_status Get_Door_status(void);
#endif
```

Light_switch.h APIs and Typedefs:

```c
#ifndef LIGHT_SWITCH_H
#define LIGHT_SWITCH_H

/**
*@brief Light switch  status
**/
typedef enum
{
    Switch_closed,
    Switch_opened
}Switch_status;

/**
*@brief Door Sensor APIs
**/

void Switch_init(void);
void Switch_Deinit(void);

Switch_status Get_Switch_status(void);
#endif
```

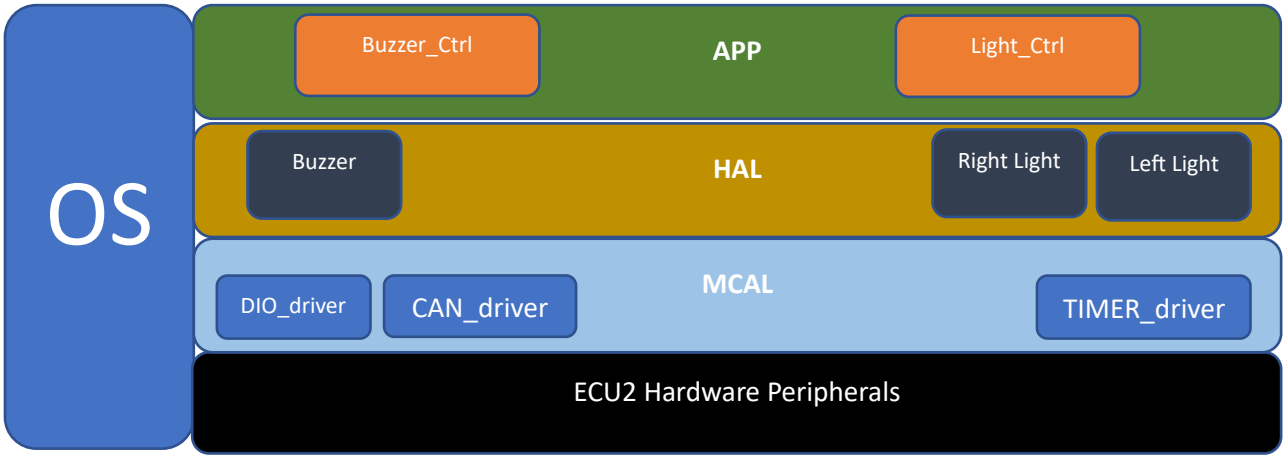
Speed_Sensor.h APIs and Typedefs:

```c
#ifndef SPEED_SENSOR_H
#define SPEED_SENSOR_H

/**
*@brief Speed Sensor status
**/
typedef enum
{
    Car_stopped,
    Car_moving
}Car_status;

/**
*@brief Speed Sensor APIs
**/

void Speed_init(void);
void Speed_Deinit(void);

Car_status Get_SpSensor_status(void);
#endif
```

Static Design for ECU2

For ECU2:

- The Layered Architecture



- ECU2 Modules
  1- CAN Module
  2- Timer Module
  3- DIO Module

- Components attached to ECU2.
  1- Buzzer
  2- Two Lights (right and left)

Buzzer Module APIs and Typedefs:

```c
#ifndef BUZZER_CTRL_H
#define BUZZER_CTRL_H

/**
*@brief Buzzer State
**/
typedef enum
{
    BUZZER_OFF,
    BUZZER_ON
}Buzzer_status;

/**
*@brief Buzzer APIs
**/

void Buzzer_init(void);
void Buzzer_Deinit(void);

Buzzer_status Get_Buzzer_status(void);
void Set_Buzzer(void);
void Reset_Buzzer(void);
#endif
```

Lights Module APIs and Typedefs:

```c
#ifndef LIGHTS_CTRL_H
#define LIGHTS_CTRL_H

/**
*@brief Lights State
**/
typedef enum
{
    Lights_OFF,
    Lights_ON
}Light_status;

/**
*@brief Lights APIs
**/

void Light_init(void);
void Light_Deinit(void);

Light_status Get_Light_status(void);
void Set_Light(void);
void Reset_Light(void);
#endif
```