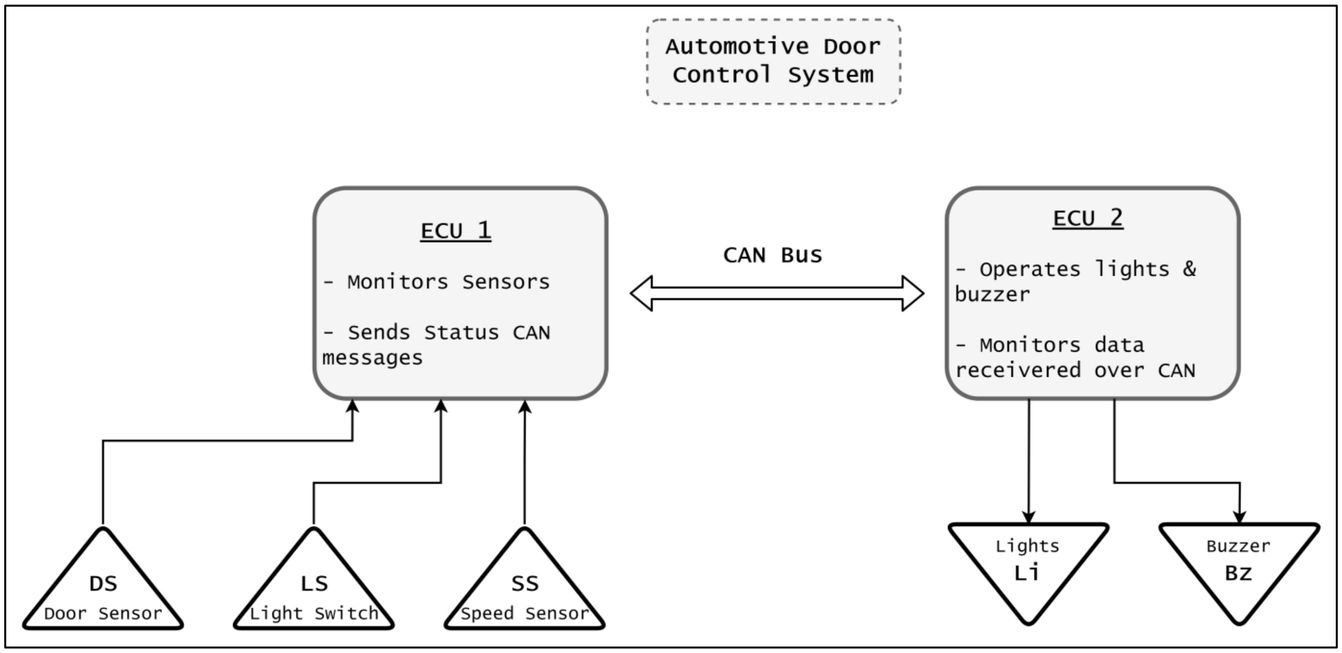
Automotive Door Control System
Static Design

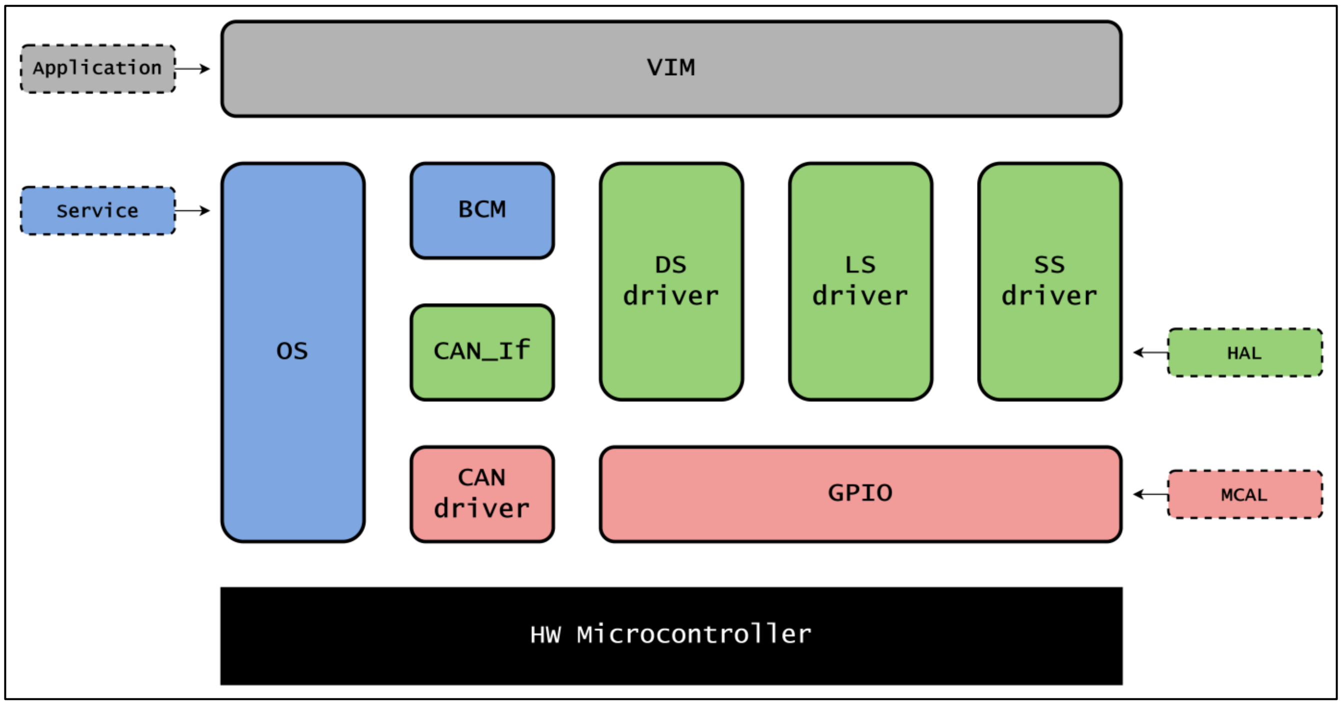
Owner: Mohamed Hossam , Email: mohamed.hossam.1183@gmail.com

1 System Schematic

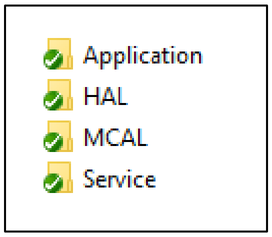

Automotive Door Control System
ECU 1
- Monitors Sensors
- Sends Status CAN messages
CAN Bus
ECU 2
- Operates lights & buzzer
- Monitors data receivered over CAN
DS
Door Sensor
LS
Light Switch
SS
Speed Sensor
Lights
Li
Buzzer
Bz

2 ECU_1

2.1 Layered Architecture:



2.2 Folder Structure



2.3 ECU Modules

- OS: Operating System: Configures and operates tasks in the system.
- BCM: Basic Communication Manager: Maintain signals for various communication protocols.
- VIM: Vehicle Infotainment Monitor: Monitor a group of sensors in the vehicle.
- DS_drv: Door Sensor Driver: Abstract DS data monitoring.
- LS_drv: Light Sensor Driver: Abstract LS data monitoring.
- SS_drv: Speed Sensor Driver: Abstract SS data monitoring.
- GPIO: General Purpose Input/Outputs: Configure and interact with IO registers.
- CAN_If: CAN Interface: Abstract CAN frame composition.
- CAN_drv: CAN driver: Configure and interact with CAN transceiver registers.

2.4 Detailed APIs:

2.4.1 OS: Operating System:

Configures and operates tasks in the system.

OS_Init

Syntax : void OS_Init(void)

Description : Initialize OS module and configure timers.

Parameters (in) : None

Parameters (out): None

Return value : None

OS_Task_Create_periodic

Syntax : Std_Result OS_Task_Create_periodic(TaskHandle_t * Task_Handler, uint8 Task_Stack_Size, uint8 Task_Period, void (* Task_Body_callback_fun)(void))

Description : create periodic task.

Parameters (in) : TaskHandle_t * Task_Handler : handler used to interact with task.

Parameters (in) : uint8 Task_Stack_Size : stack memory to be allocated to task.

Parameters (in) : uint8 Task_Period : desired task periodicity.

Parameters (in) : void (* Task_Body_callback_fun)(void) : pointer to function containing task body.

Parameters (out): None

Return value : Std_Result (OK / NOT_OK)

```
OS_Task_Delete

Syntax          : Std_Result OS_Task_Delete( TaskHandle_t
Task_Handler)

Description     : delete task.

Parameters (in) : TaskHandle_t Task_Handler : handler of desired
task.

Parameters (out): None

Return value    : Std_Result ( OK / NOT_OK )


OS_StartScheduler

Syntax          : void OS_StartScheduler( void )

Description     : Start scheduling and dispatching tasks.

Parameters (in) : None

Parameters (out): None

Return value    : None
```

2.4.2 BCM: Basic Communication Manager:

Maintain signals for various communication protocols.

BCM_Init

Syntax : void BCM_Init(void)

Description : Initialize BCM module.

Parameters (in) : None

Parameters (out): None

Return value : None

BCM_Create_Signal

Syntax : Std_Result BCM_Create_Signal(SignalHandle_t * Signal_Handler, COM_Protocol_t cp_type, Dir_t Direction, uint8 Signal_Periodicity, void (* Signal_Updater_callback_fun), uint8 Payload_Size)

Description : create and configure new signal.

Parameters (in) : SignalHandle_t * Signal_Handler: handler used to interact with signal.

Parameters (in) : COM_Protocol_t cp_type: enum to define used communication protocol.

Parameters (in) : Dir_t Direction (Input / Output).

Parameters (in) : uint8 Signal_Periodicity: desired signal periodicity.

Parameters (in) : void (* Signal_Updater_callback_fun): pointer to function containing code to update or fetch signal value.

Parameters (in) : uint8 Payload_Size: signal payload size.

Parameters (out): None

Return value : Std_Result (OK / NOT_OK)

BCM_Send_Signal

Syntax : Std_Result BCM_Send_Signal(uint8 Signal_Handler, (uint32 * Payload_Data))

Description : Send signal.

Parameters (in) : uint8 Signal_Handler: handler of desired signal.

```
Parameters (in) : (uint32 * Payload_Data): signal payload size.

Parameters (out): None

Return value    : Std_Result ( OK / NOT_OK )


BCM_Receive_Signal

Syntax          : Std_Result BCM_Receive_Signal(uint8
Signal_Handler, (uint32 * Payload_Data))

Description     : Receive signal.

Parameters (in) : uint8 Signal_Handler: handler of desired signal.

Parameters (in) : (uint32 * Payload_Data): signal payload size.

Parameters (out): None

Return value    : Std_Result ( OK / NOT_OK )
```

2.4.3 VIM: Vehicle Infotainment Monitor:

Monitor a group of sensors in the vehicle.

```
VIM_Init
Syntax          : void VIM_Init( void )
Description     : Initialize VIM module.
Parameters (in) : None
Parameters (out): None
Return value    : None

VIM_DS_Callback
Syntax          : void VIM_DS_Callback(uint32 * status_Data)
Description     : door sensor status callback.
Parameters (in) : uint32 * status_Data: sensor status data
Parameters (out): None
Return value    : None

VIM_LS_Callback
Syntax          : void VIM_LS_Callback(uint32 * status_Data)
Description     : light switch sensor status callback.
Parameters (in) : uint32 * status_Data: sensor status data
Parameters (out): None
Return value    : None

VIM_SS_Callback
Syntax          : void VIM_SS_Callback(uint32 * status_Data)
Description     : speed sensor status callback.
Parameters (in) : uint32 * status_Data: sensor status data
Parameters (out): None
Return value    : None
```

2.4.4 DS_drv: Door Sensor Driver:

Abstract DS data monitoring.

```
DS_Init
Syntax          : void DS_Init(void)
Description     : Initialize DS_drv module.
Parameters (in) : None
Parameters (out): None
Return value    : None

DS_Get_Data
Syntax          : void DS_Get_Data(uint32 * Data)
Description     : Get door sensor status.
Parameters (in) : uint32 * Data: sensor status data
Parameters (out): None
Return value    : None
```

2.4.5 LS_drv: Light switch Sensor Driver:

Abstract LS data monitoring.

```
LS_Init
Syntax          : void LS_Init(void)
Description     : Initialize LS_drv module.
Parameters (in) : None
Parameters (out): None
Return value    : None
```

```
LS_Get_Data
Syntax          : void LS_Get_Data(uint32 * Data)
Description     : Get light switch sensor status.
Parameters (in) : uint32 * Data: sensor status data
Parameters (out): None
Return value    : None
```

2.4.6 SS_drv: Speed Sensor Driver:

Abstract SS data monitoring.

```
SS_Init
Syntax           : void SS_Init(void)
Description      : Initialize SS_drv module.
Parameters (in)  : None
Parameters (out) : None
Return value     : None

SS_Get_Data
Syntax           : void SS_Get_Data(uint32 * Data)
Description      : Get speed sensor status.
Parameters (in)  : uint32 * Data: sensor status data
Parameters (out) : None
Return value     : None
```

2.4.7 GPIO: General Purpose Input/Outputs:

Configure and interact with IO registers.

the label "[XX]" denotes the different I/O registers belonging to various sensors (door , light switch, and speed), and various warning outputs (buzzer , and lights).

```
Reg[XX]_Cfg
Syntax           : void Reg[XX]_Cfg(Dir_t Direction)
Description      : Configure register [XX] as intput or output.
Parameters (in)  : Dir_t Direction ( Input / Output ).
Parameters (out): None
Return value     : None
```

```
Reg[XX]_Read
Syntax           : void Reg[XX]_Read(uint32 * Data)
Description      : Read register [XX] value.
Parameters (in)  : uint32 * Data: payload data.
Parameters (out): None
Return value     : None
```

```
Reg[XX]_Write
Syntax           : void Reg[XX]_Write(uint32 * Data)
Description      : write register [XX] value.
Parameters (in)  : uint32 * Data: payload data.
Parameters (out): None
Return value     : None
```

2.4.8 CAN_If: CAN Interface:

Abstract CAN frame composition.

CAN_If_Init

Syntax : void CAN_If_Init(void)

Description : Initialize CAN_If module.

Parameters (in) : None

Parameters (out): None

Return value : None

CAN_If_Create_Message

Syntax : void CAN_If_Create_Message(MessageHandle_t * Message_Handler)

Description : create new CAN message.

Parameters (in) : MessageHandle_t * Message_Handler: handler used to interact with message.

Parameters (out): None

Return value : None

CAN_If_Create_Signal

Syntax : void CAN_If_Create_Signal(SignalHandle_t * Signal_Handler)

Description : create new CAN Signal.

Parameters (in) : SignalHandle_t * Signal_Handler: handler used to interact with signal.

Parameters (out): None

Return value : None

CAN_If_Cfg_Signal

Syntax   : void CAN_If_Cfg_Signal(MessageHandle_t * Message_Handler , SignalHandle_t * Signal_Handler)

Description   : Configure CAN signal inside CAN frame.

Parameters (in) : MessageHandle_t * Message_Handler: handler of desired message.

Parameters (in) : SignalHandle_t * Signal_Handler: handler of desired signal.

Parameters (out): None

Return value   : None

CAN_If_Transmit

Syntax   : Std_Result CAN_If_Transmit(SignalHandle_t * Signal_Handler, uint8 * Payload_Data, uint8 Payload_Size)

Description   : Transmit signal.

Parameters (in) : SignalHandle_t * Signal_Handler: handler of desired signal.

Parameters (in) : uint8 * Payload_Data: frame data

Parameters (in) : uint8 Payload_Size: payload data size.

Parameters (out): None

Return value   : Std_Result (OK / NOT_OK)

CAN_If_Receive

Syntax   : Std_Result CAN_If_Receive(SignalHandle_t * Signal_Handler, uint8 * Payload_Data, uint8 Payload_Size)

Description   : Receive signal.

Parameters (in) : SignalHandle_t * Signal_Handler: handler of desired signal.

Parameters (in) : uint8 * Payload_Data: frame data

Parameters (in) : uint8 Payload_Size: payload data size.

Parameters (out): None

Return value   : Std_Result (OK / NOT_OK)

2.4.9 CAN_drv: CAN driver:

Configure and interact with CAN transceiver registers.

CAN_Init

```
Syntax          : void CAN_Init(void)
Description     : Initialize CAN_drv module.
Parameters (in) : None
Parameters (out): None
Return value    : None
```

CAN_Set_Baudrate

```
Syntax          : Std_Result CAN_Set_Baudrate(uint32 Rate)
Description     : Set CAN baud rate.
Parameters (in) : uint32 Rate: baud rate value.
Parameters (out): None
Return value    : Std_Result ( OK / NOT_OK )
```

CAN_Write

```
Syntax          : Std_Result CAN_Write(uint8 * Data, uint8 BuffSize)
Description     : Write frame data to registers.
Parameters (in) : uint8 * Data: frame data
Parameters (in) : uint8 BuffSize: data size.
Parameters (out): None
Return value    : Std_Result ( OK / NOT_OK )
```

CAN_Read

```
Syntax          : Std_Result CAN_Read(uint8 * Data, uint8 BuffSize)
Description     : Read frame data from registers.
Parameters (in) : uint8 * Data: frame data
Parameters (in) : uint8 BuffSize: data size.
Parameters (out): None
Return value    : Std_Result ( OK / NOT_OK )
```

2.4.10 Typedefs:

```
typedef enum
{
  CAN = 0,
  CAN_FD,
  Ethernet,
  Flexray,
  Lin
}COM_Protocol_t;

typedef enum
{
  NOT_OK = 0,
  OK
}Std_Result;

typedef enum
{
  Input = 0,
  Output
}Dir_t
```

```
typedef unsigned long int TaskHandle_t;
```

Type by which tasks are referenced, that can then be used as a parameter to functions that interact with tasks.

```
typedef unsigned long int MessageHandle_t;
```

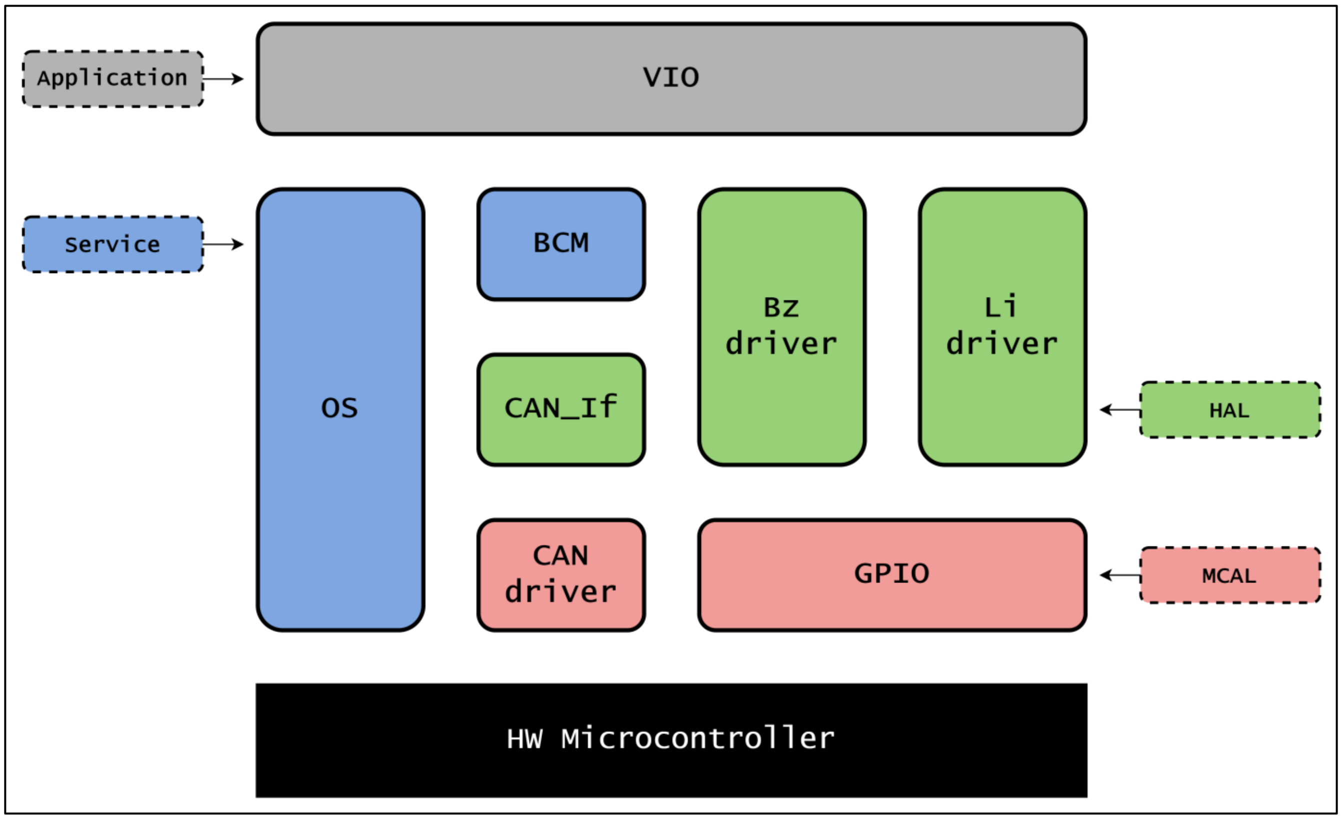
Type by which messages are referenced, that can then be used as a parameter to functions that interact with messages.

```
typedef unsigned long int SignalHandle_t;
```

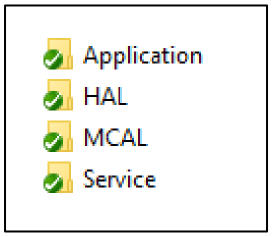
Type by which signals are referenced, that can then be used as a parameter to functions that interact with signals.

3 ECU_2

3.1 Layered Architecture:



3.2 Folder Structure


Application
HAL
MCAL
Service

3.3 ECU Modules

- OS: Operating System: Configures and operates tasks in the system.
- BCM: Basic Communication Manager: Maintain signals for various communication protocols.
- VIO: Vehicle Infotainment Operator: Operates a group of audio visual warning signal in the vehicle.
- Bz_drv: Buzzer driver: Abstract Buzzer operation.
- Li_drv: Lights driver: Abstract Lights operation.
- GPIO: General Purpose Input/Outputs: Configure and interact with IO registers.
- CAN_If: CAN Interface: Abstract CAN frame composition.
- CAN_drv: CAN driver: Configure and interact with CAN transceiver registers.

3.4 Detailed APIs:

3.4.1 OS: Operating System:

Configures and operates tasks in the system.

OS_Init

Syntax : void OS_Init(void)

Description : Initialize OS module and configure timers.

Parameters (in) : None

Parameters (out): None

Return value : None

OS_Task_Create_periodic

Syntax : Std_Result OS_Task_Create_periodic(TaskHandle_t * Task_Handler, uint8 Task_Stack_Size, uint8 Task_Period, void (* Task_Body_callback_fun)(void))

Description : create periodic task.

Parameters (in) : TaskHandle_t * Task_Handler : handler used to interact with task.

Parameters (in) : uint8 Task_Stack_Size : stack memory to be allocated to task.

Parameters (in) : uint8 Task_Period : desired task periodicity.

Parameters (in) : void (* Task_Body_callback_fun)(void) : pointer to function containing task body.

Parameters (out): None

Return value : Std_Result (OK / NOT_OK)

OS_Task_Delete

Syntax : Std_Result OS_Task_Delete(TaskHandle_t Task_Handler)

Description : delete task.

Parameters (in) : TaskHandle_t Task_Handler : handler of desired task.

Parameters (out): None

```
Return value    : Std_Result ( OK / NOT_OK )

OS_StartScheduler
Syntax          : void OS_StartScheduler( void )
Description     : Start scheduling and dispatching tasks.
Parameters (in) : None
Parameters (out): None
Return value    : None
```

3.4.2 BCM: Basic Communication Manager:

Maintain signals for various communication protocols.

BCM_Init

Syntax   : void BCM_Init(void)

Description   : Initialize BCM module.

Parameters (in) : None

Parameters (out) : None

Return value   : None

BCM_Create_Signal

Syntax   : Std_Result BCM_Create_Signal(SignalHandle_t * Signal_Handler, COM_Protocol_t cp_type, Dir_t Direction, uint8 Signal_Periodicity, void (* Signal_Updater_callback_fun), uint8 Payload_Size)

Description   : create and configure new signal.

Parameters (in) : SignalHandle_t * Signal_Handler: handler used to interact with signal.

Parameters (in) : COM_Protocol_t cp_type: enum to define used communication protocol.

Parameters (in) : Dir_t Direction (Input / Output).

Parameters (in) : uint8 Signal_Periodicity: desired signal periodicity.

Parameters (in) : void (* Signal_Updater_callback_fun): pointer to function containing code to update or fetch signal value.

Parameters (in) : uint8 Payload_Size: signal payload size.

Parameters (out) : None

Return value   : Std_Result (OK / NOT_OK)

BCM_Send_Signal

Syntax   : Std_Result BCM_Send_Signal(uint8 Signal_Handler, (uint32 * Payload_Data))

Description   : Send signal.

Parameters (in) : uint8 Signal_Handler: handler of desired signal.

```
Parameters (in) : (uint32 * Payload_Data): signal payload size.
Parameters (out): None
Return value    : Std_Result ( OK / NOT_OK )
```

```
BCM_Receive_Signal
Syntax          : Std_Result BCM_Receive_Signal(uint8
Signal_Handler, (uint32 * Payload_Data))
Description     : Receive signal.
Parameters (in) : uint8 Signal_Handler: handler of desired signal.
Parameters (in) : (uint32 * Payload_Data): signal payload size.
Parameters (out): None
Return value    : Std_Result ( OK / NOT_OK )
```

3.4.3 VIO: Vehicle Infotainment Operator:

Operates a group of audio visual warning signal in the vehicle.

```
VIO_Init
Syntax          : void VIO_Init(void)
Description     : Initialize VIO module.
Parameters (in) : None
Parameters (out): None
Return value    : None

void VIO_DS_Update(uint32 * status_Data)
Syntax          : void VIO_DS_Update(uint32 * status_Data)
Description     : door sensor status update local value.
Parameters (in) : uint32 * status_Data: sensor status data.
Parameters (out): None
Return value    : None

void VIO_LS_Update(uint32 * status_Data)
Syntax          : void VIO_LS_Update(uint32 * status_Data)
Description     : light switch sensor status update local value.
Parameters (in) : uint32 * status_Data: sensor status data.
Parameters (out): None
Return value    : None

void VIO_SS_Update(uint32 * status_Data)
Syntax          : void VIO_SS_Update(uint32 * status_Data)
Description     : speed sensor status update local value.
Parameters (in) : uint32 * status_Data: sensor status data.
Parameters (out): None
Return value    : None
```

VIO_Main

Syntax   : void VIO_Main(void)

Description   : Periodicaly evaluate sensor values and operate warnings accordingly.

Parameters (in) : None

Parameters (out): None

Return value   : None

3.4.4 Bz_drv: Buzzer driver:

Abstract Buzzer operation.

```
Bz_Init
Syntax          : void Bz_Init(void)
Description     : Initialize Bz_drv module.
Parameters (in) : None
Parameters (out): None
Return value    : None

Bz_Set_Data
Syntax          : void Bz_Set_Data(uint32 * Data)
Description     : Set Buzzer ON or OFF.
Parameters (in) : uint32 * Data: status data
Parameters (out): None
Return value    : None
```

3.4.5 Li_drv: Lights driver:

```
Abstract Lights operation.

Li_Init
Syntax          : void Li_Init(void)
Description     : Initialize Li_drv module.
Parameters (in) : None
Parameters (out): None
Return value    : None

Li_Set_Data
Syntax          : void Li_Set_Data(uint32 * Data)
Description     : Set Lights ON or OFF.
Parameters (in) : uint32 * Data: status data
Parameters (out): None
Return value    : None
```

3.4.6 GPIO: General Purpose Input/Outputs:

Configure and interact with IO registers.

the label "[XX]" denotes the different I/O registers belonging to various sensors (door , light switch, and speed)

, and various warning outputs (buzzer , and lights).

```
Reg[XX]_Cfg
Syntax           : void Reg[XX]_Cfg(Dir_t Direction)
Description      : Configure register [XX] as intput or output.
Parameters (in)  : Dir_t Direction ( Input / Output ).
Parameters (out): None
Return value     : None

Reg[XX]_Read
Syntax           : void Reg[XX]_Read(uint32 * Data)
Description      : Read register [XX] value.
Parameters (in)  : uint32 * Data: payload data.
Parameters (out): None
Return value     : None

Reg[XX]_Write
Syntax           : void Reg[XX]_Write(uint32 * Data)
Description      : write register [XX] value.
Parameters (in)  : uint32 * Data: payload data.
Parameters (out): None
Return value     : None
```

3.4.7 CAN_If: CAN Interface:

Abstract CAN frame composition.

CAN_If_Init

Syntax : void CAN_If_Init(void)

Description : Initialize CAN_If module.

Parameters (in) : None

Parameters (out): None

Return value : None

CAN_If_Create_Message

Syntax : void CAN_If_Create_Message(MessageHandle_t * Message_Handler)

Description : create new CAN message.

Parameters (in) : MessageHandle_t * Message_Handler: handler used to interact with message.

Parameters (out): None

Return value : None

CAN_If_Create_Signal

Syntax : void CAN_If_Create_Signal(SignalHandle_t * Signal_Handler)

Description : create new CAN Signal.

Parameters (in) : SignalHandle_t * Signal_Handler: handler used to interact with signal.

Parameters (out): None

Return value : None

CAN_If_Cfg_Signal

Syntax : void CAN_If_Cfg_Signal(MessageHandle_t * Message_Handler , SignalHandle_t * Signal_Handler)

Description : Configure CAN signal inside CAN frame.

Parameters (in) : MessageHandle_t * Message_Handler: handler of desired message.

Parameters (in) : SignalHandle_t * Signal_Handler: handler of desired signal.

Parameters (out): None

Return value   : None

CAN_If_Transmit

Syntax   : Std_Result CAN_If_Transmit(SignalHandle_t * Signal_Handler, uint8 * Payload_Data, uint8 Payload_Size)

Description   : Transmit signal.

Parameters (in) : SignalHandle_t * Signal_Handler: handler of desired signal.

Parameters (in) : uint8 * Payload_Data: frame data

Parameters (in) : uint8 Payload_Size: payload data size.

Parameters (out): None

Return value   : Std_Result (OK / NOT_OK)

CAN_If_Receive

Syntax   : Std_Result CAN_If_Receive(SignalHandle_t * Signal_Handler, uint8 * Payload_Data, uint8 Payload_Size)

Description   : Receive signal.

Parameters (in) : SignalHandle_t * Signal_Handler: handler of desired signal.

Parameters (in) : uint8 * Payload_Data: frame data

Parameters (in) : uint8 Payload_Size: payload data size.

Parameters (out): None

Return value   : Std_Result (OK / NOT_OK)

3.4.8 CAN_drv: CAN driver:

```
Configure and interact with CAN transceiver registers.
CAN_Init
Syntax          : void CAN_Init(void)
Description     : Initialize CAN_drv module.
Parameters (in) : None
Parameters (out): None
Return value    : None

CAN_Set_Baudrate
Syntax          : Std_Result CAN_Set_Baudrate(uint32 Rate)
Description     : Set CAN baud rate.
Parameters (in) : uint32 Rate: baud rate value.
Parameters (out): None
Return value    : Std_Result ( OK / NOT_OK )

CAN_Write
Syntax          : Std_Result CAN_Write(uint8 * Data, uint8 BuffSize)
Description     : Write frame data to registers.
Parameters (in) : uint8 * Data: frame data
Parameters (in) : uint8 BuffSize: data size.
Parameters (out): None
Return value    : Std_Result ( OK / NOT_OK )

CAN_Read
Syntax          : Std_Result CAN_Read(uint8 * Data, uint8 BuffSize)
Description     : Read frame data from registers.
Parameters (in) : uint8 * Data: frame data
Parameters (in) : uint8 BuffSize: data size.
Parameters (out): None
Return value    : Std_Result ( OK / NOT_OK )
```

3.4.9 Typedefs:

```
typedef enum
{
  CAN = 0,
  CAN_FD,
  Ethernet,
  Flexray,
  Lin
}COM_Protocol_t;

typedef enum
{
  NOT_OK = 0,
  OK
}Std_Result;

typedef enum
{
  Input = 0,
  Output
}Dir_t
```

```
typedef unsigned long int TaskHandle_t;
```

Type by which tasks are referenced, that can then be used as a parameter to functions that interact with tasks.

```
typedef unsigned long int MessageHandle_t;
```

Type by which messages are referenced, that can then be used as a parameter to functions that interact with messages.

```
typedef unsigned long int SignalHandle_t;
```

Type by which signals are referenced, that can then be used as a parameter to functions that interact with signals.