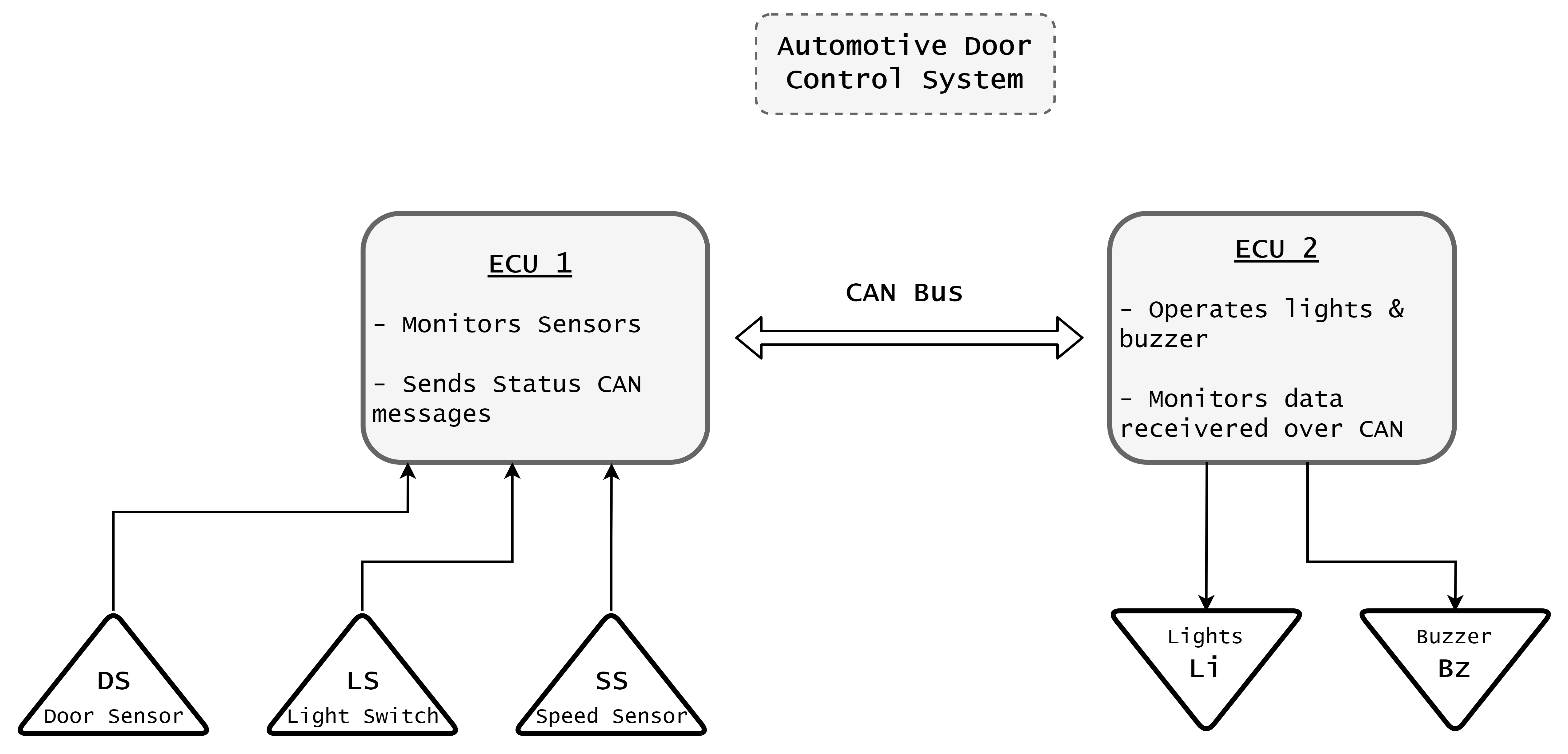
*Automotive Door Control System*

Static Design

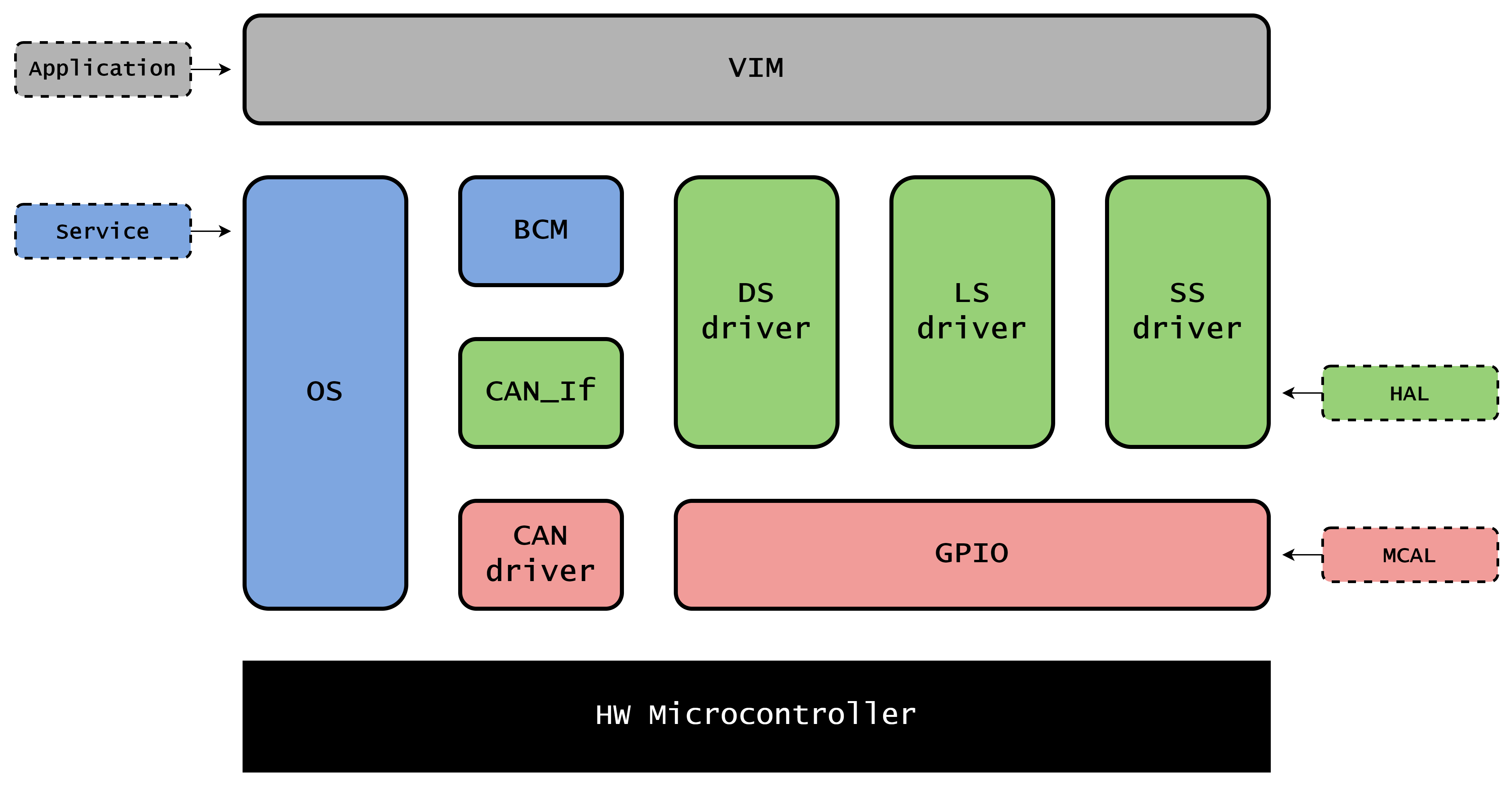
Owner: Mohamed Hossam , Email: [mohamed.hossam.1183@gmail.com](mailto:mohamed.hossam.1183@gmail.com)

# System Schematic

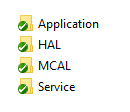


# ECU\_1

## Layered Architecture:



## Folder Structure



## 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.

## Detailed APIs:

### 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

### 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 )

### 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

### 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

### 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

### 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

### 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

### 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 )

### 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 )

### 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;

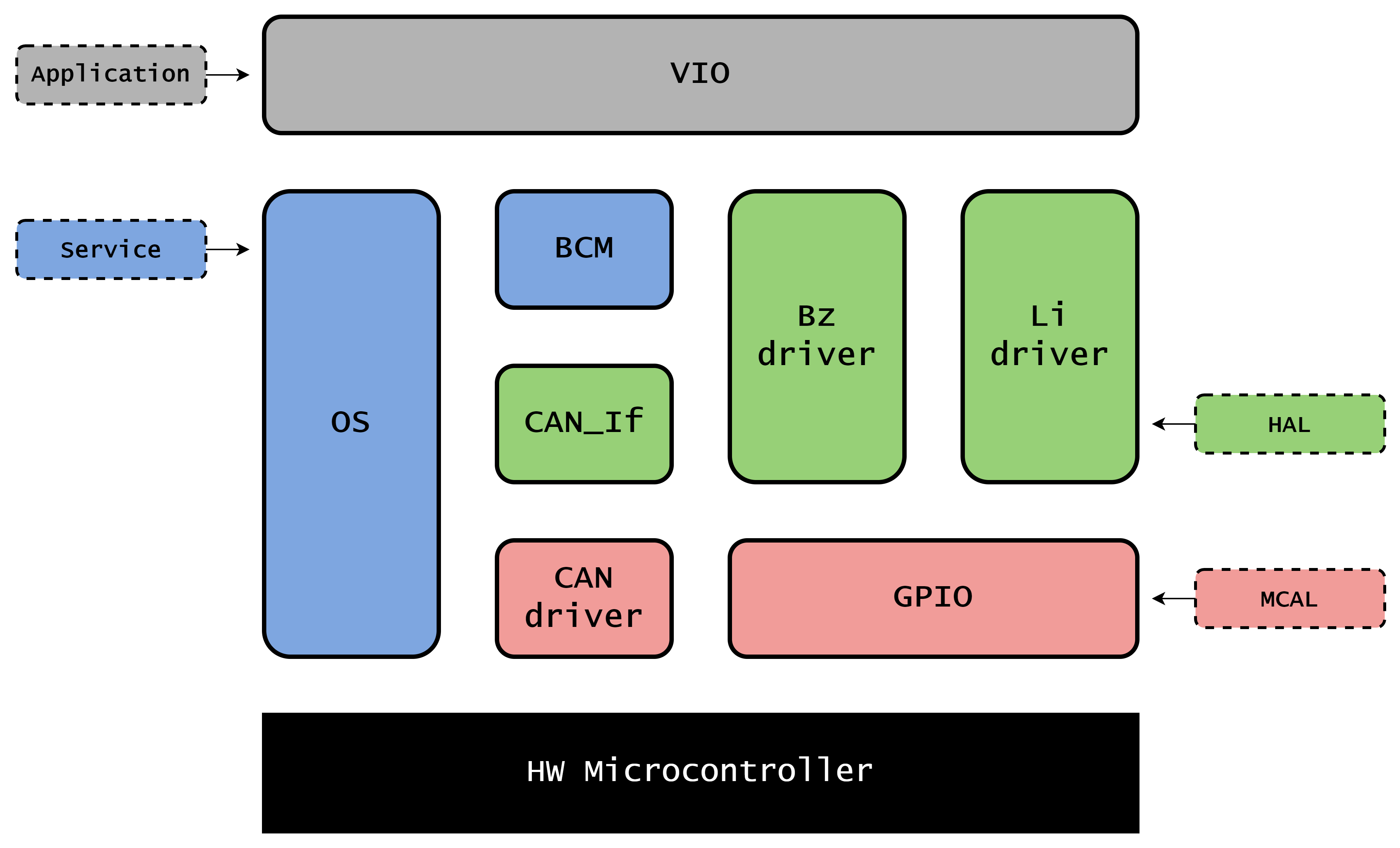
Type by which messages are referenced, that can then be used as a parameter to functions that interact with messages.

typedef unsigned long int SignalHandle\_t;

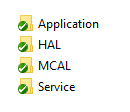
Type by which signals are referenced, that can then be used as a parameter to functions that interact with signals.

# ECU\_2

## Layered Architecture:



## Folder Structure



## 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.

## Detailed APIs:

### 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

### 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 )

### 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

### 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

### 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

### 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

### 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 )

### 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 )

### 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.