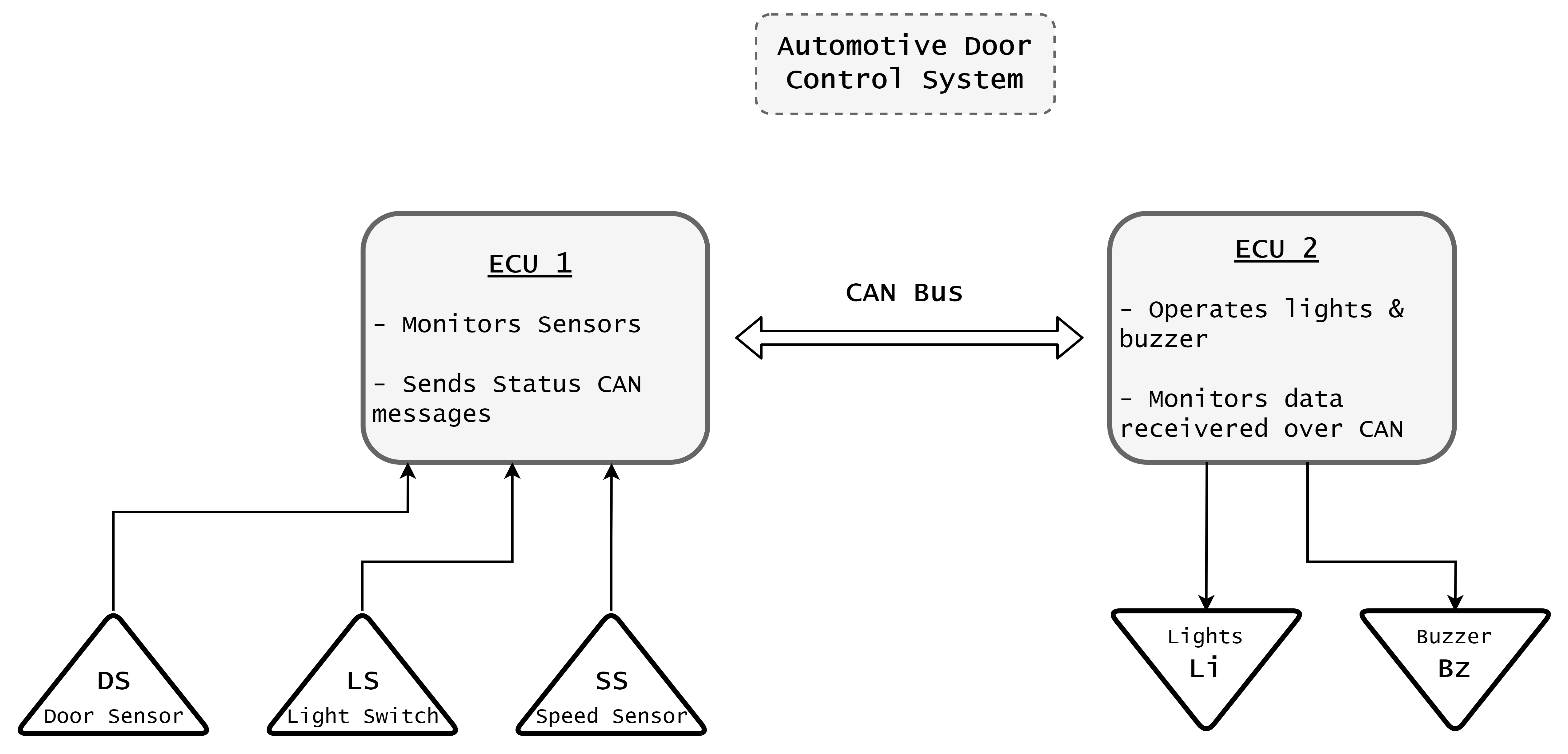
*Automotive Door Control System*

Static Design

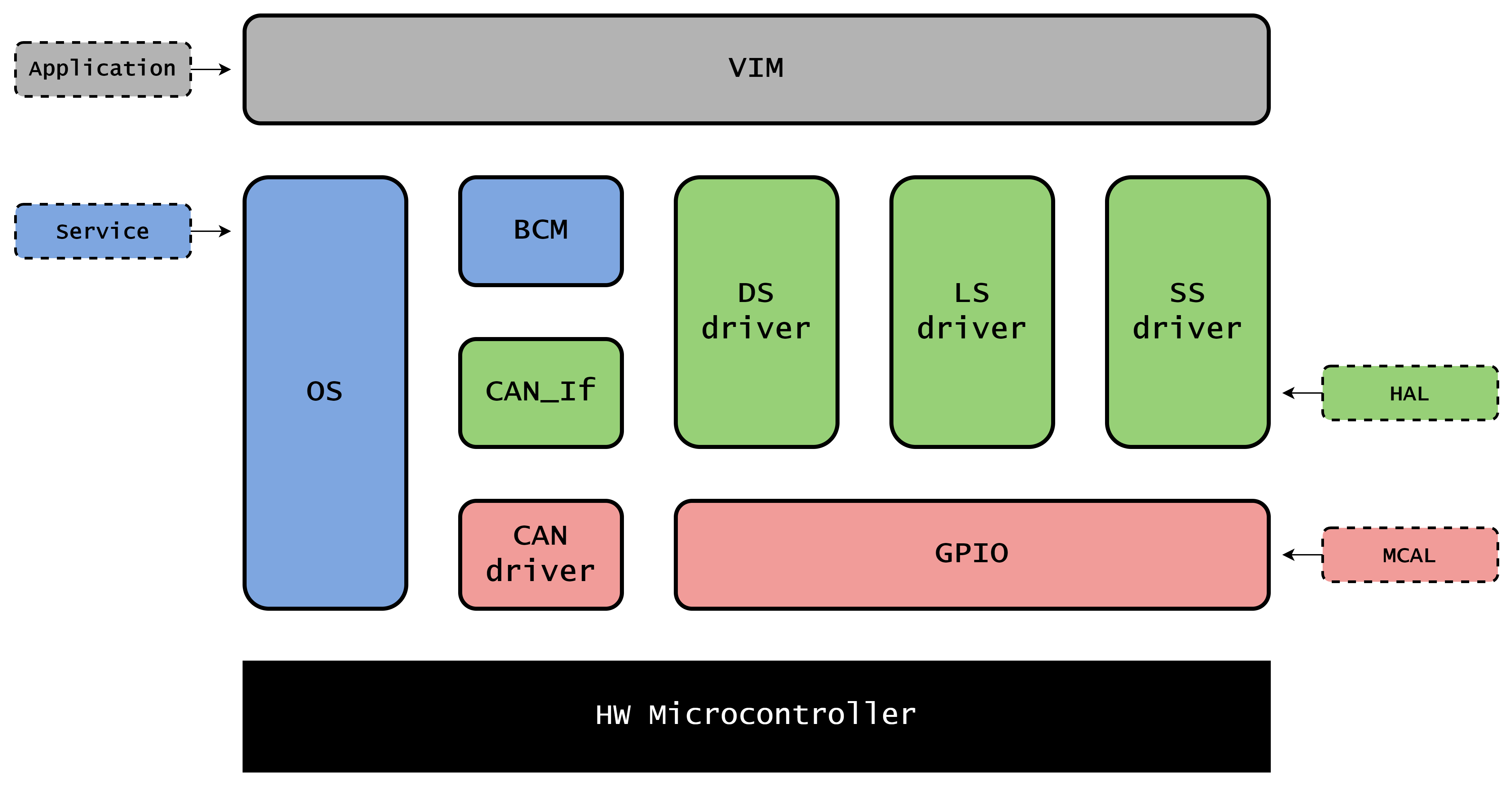
Owner: Mohamed Hossam , Email: [mohamed.hossam.1183@gmail.com](mailto:mohamed.hossam.1183@gmail.com)

# System Schematic

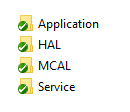


# ECU\_1

## 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:

- void OS\_Init( void ): Initialize OS module and configure timers.

- Std\_Result OS\_Task\_Create\_periodic( uint8 \* Task\_Handler,  
uint8 Task\_Stack\_Size,  
uint8 Task\_Period,  
void (\* Task\_Body\_callback\_fun)(void) ): create periodic task.

- Std\_Result OS\_Task\_Delete( uint8 Task\_Handler): delete task.

- void OS\_StartScheduler( void ): Start scheduling and dispatching tasks.

### BCM: Basic Communication Manager:

- void BCM\_Init( void ): Initialize BCM module.

- Std\_Result BCM\_Create\_Signal( uint8 \* Signal\_Handler, COM\_Protocol\_t cp\_type,  
Dir\_t Direction,  
uint8 Signal\_Periodicity,  
void (\* Signal\_Updater\_callback\_fun),   
uint8 Payload\_Size): create and configure new signal.

- Std\_Result BCM\_Send\_Signal(uint8 Signal\_Handler,   
(uint32 \* Payload\_Data)): Send signal.

- Std\_Result BCM\_Receive\_Signal(uint8 Signal\_Handler,   
(uint32 \* Payload\_Data)): Receive signal.

### VIM: Vehicle Infotainment Monitor:

- void VIM\_Init(void): Initialize VIM module.

- void VIM\_DS\_Callback(uint32 \* status\_Data): door sensor status callback.

- void VIM\_LS\_Callback(uint32 \* status\_Data): light sensor status callback.

- void VIM\_SS\_Callback(uint32 \* status\_Data): speed sensor status callback.

### DS\_drv: Door Sensor Driver:

- void DS\_Init(void): Initialize DS\_drv module.

- void DS\_Get\_Data(uint32 \* Data): Get door sensor status.

### LS\_drv: Light Sensor Driver:

- void LS\_Init(void): Initialize LS\_drv module.

- void LS\_Get\_Data(uint32 \* Data): Get light sensor status.

### SS\_drv: Speed Sensor Driver:

- void SS\_Init(void): Initialize SS\_drv module.

- void SS\_Get\_Data(uint32 \* Data): Get speed sensor status.

### GPIO: General Purpose Input/Outputs:

- void Reg[XX]\_Cfg(Dir\_t Direction): Configure register [XX] as intput or output.

- void Reg[XX]\_Read(uint32 \* Data): Read register [XX] value.

- void Reg[XX]\_Write(uint32 \* Data): write register [XX] value.

### CAN\_If: CAN Interface:

- void CAN\_If\_Init(void): Initialize CAN\_If module.

- void CAN\_If\_Create\_Message(uint8 \* Message\_Handler): create new CAN message.

- void CAN\_If\_Create\_Signal(uint8 \* Signal\_Handler): create new CAN Signal.

- void CAN\_If\_Cfg\_Signal(uint8 \* Message\_Handler ,   
uint8 \* Signal\_Handler): Configure CAN signal inside CAN frame.

- Std\_Result CAN\_If\_Transmit(uint8 \* Signal\_Handler,   
uint8 \* Payload\_Data,   
uint8 Payload\_Size): transmit signal.

- Std\_Result CAN\_If\_Receive(uint8 \* Signal\_Handler,   
uint8 \* Payload\_Data,   
uint8 Payload\_Size): Receive signal.

### CAN\_drv: CAN driver:

- void CAN\_Init(void): Initialize CAN\_drv module.

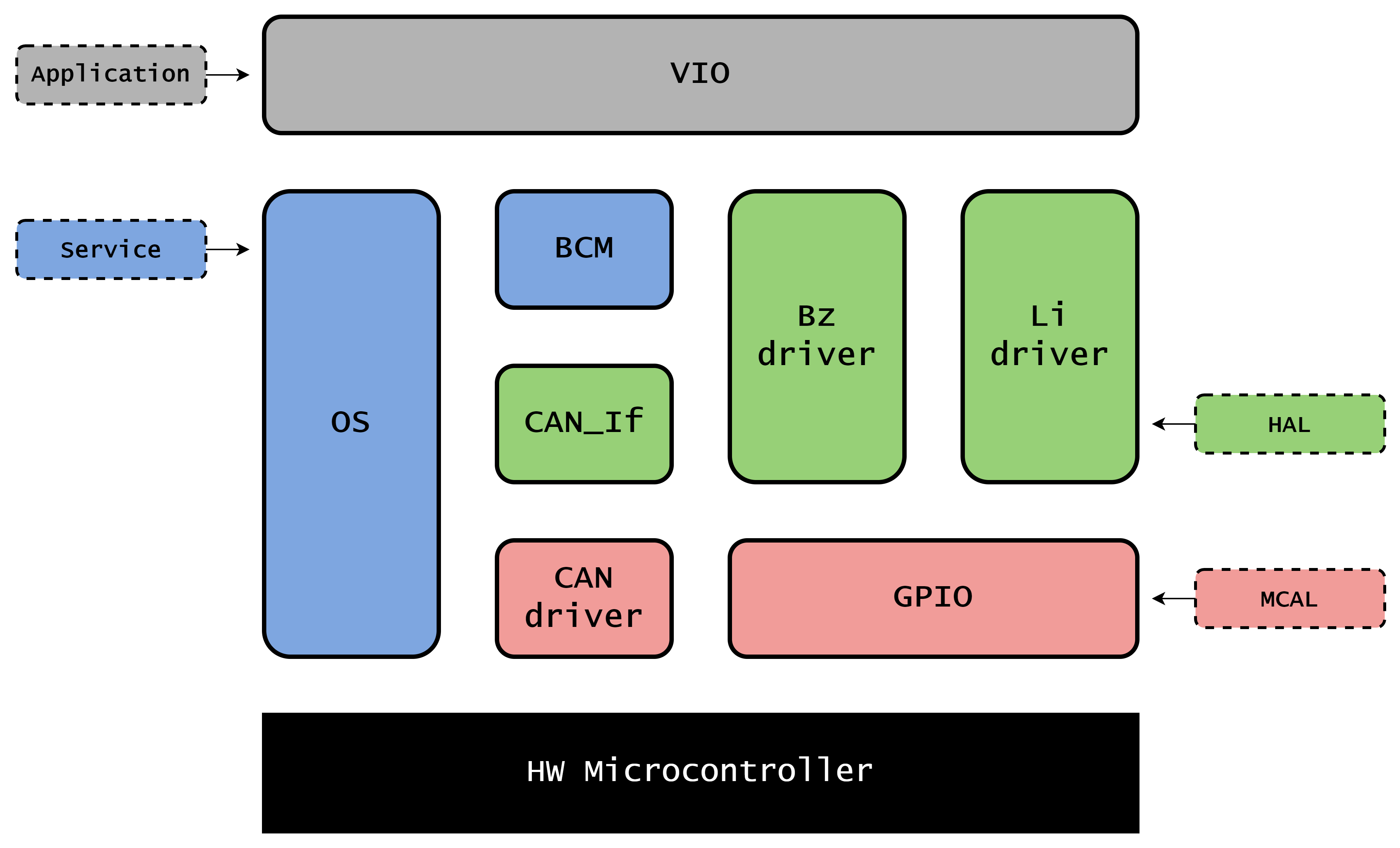
- void CAN\_Set\_Baudrate(uint32 Rate): Set CAN baud rate.

- Std\_Result CAN\_Write(uint8 \* Data,   
uint8 BuffSize): Write frame data to registers.

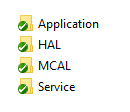
- Std\_Result CAN\_Read(uint8 \* Data,   
uint8 BuffSize): Write frame data to registers.

# 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:

- void OS\_Init( void ): Initialize OS module and configure timers.

- Std\_Result OS\_Task\_Create\_periodic( uint8 \* Task\_Handler,   
uint8 Task\_Stack\_Size,   
uint8 Task\_Period,   
void (\* Task\_Body\_callback\_fun)(void) ): create periodic task.

- Std\_Result OS\_Task\_Delete( uint8 Task\_Handler): delete task.

- void OS\_StartScheduler( void ): Start scheduling and dispatching tasks.

### BCM: Basic Communication Manager:

- void BCM\_Init( void ): Initialize BCM module.

- Std\_Result BCM\_Create\_Signal( uint8 \* Signal\_Handler, COM\_Protocol\_t cp\_type,   
Dir\_t Direction,   
uint8 Signal\_Periodicity,   
void (\* Signal\_Updater\_callback\_fun),   
uint8 Payload\_Size): create and configure new signal.

- Std\_Result BCM\_Send\_Signal(uint8 Signal\_Handler,   
(uint32 \* Payload\_Data)): Send signal.

- Std\_Result BCM\_Receive\_Signal(uint8 Signal\_Handler,   
(uint32 \* Payload\_Data)): Receive signal.

### VIO: Vehicle Infotainment Operator:

- void VIO\_Init(void): Initialize VIO module.

- void VIO\_DS\_Update(uint32 \* status\_Data): door sensor status update local value.

- void VIO\_LS\_Update(uint32 \* status\_Data): light sensor status update local value.

- void VIO\_SS\_Update(uint32 \* status\_Data): speed sensor status update local value.

- void VIO\_Main(void): Periodicaly evaluate sensor values and operate warnings accordingly.

### Bz\_drv: Buzzer driver:

- void Bz\_Init(void): Initialize Bz\_drv module.

- void Bz\_Set\_Data(uint32 \* Data): Set Buzzer ON or OFF.

### Li\_drv: Lights driver:

- void Li\_Init(void): Initialize Li\_drv module.

- void Li\_Set\_Data(uint32 \* Data): Set Lights ON or OFF.

### GPIO: General Purpose Input/Outputs:

- void Reg[XX]\_Cfg(Dir\_t Direction): Configure register [XX] as intput or output.

- void Reg[XX]\_Read(uint32 \* Data): Read register [XX] value.

- void Reg[XX]\_Write(uint32 \* Data): write register [XX] value.

### CAN\_If: CAN Interface:

- void CAN\_If\_Init(void): Initialize CAN\_If module.

- void CAN\_If\_Create\_Message(uint8 \* Message\_Handler): create new CAN message.

- void CAN\_If\_Create\_Signal(uint8 \* Signal\_Handler): create new CAN Signal.

- void CAN\_If\_Cfg\_Signal(uint8 \* Message\_Handler ,   
uint8 \* Signal\_Handler): Configure CAN signal inside CAN frame.

- Std\_Result CAN\_If\_Transmit(uint8 \* Signal\_Handler,   
uint8 \* Payload\_Data,   
uint8 Payload\_Size): transmit signal.

- Std\_Result CAN\_If\_Receive(uint8 \* Signal\_Handler,   
uint8 \* Payload\_Data,   
uint8 Payload\_Size): Receive signal.

### CAN\_drv: CAN driver:

- void CAN\_Init(void): Initialize CAN\_drv module.

- void CAN\_Set\_Baudrate(uint32 Rate): Set CAN baud rate.

- Std\_Result CAN\_Write(uint8 \* Data,   
uint8 BuffSize): Write frame data to registers.

- Std\_Result CAN\_Read(uint8 \* Data,   
uint8 BuffSize): Write frame data to registers.