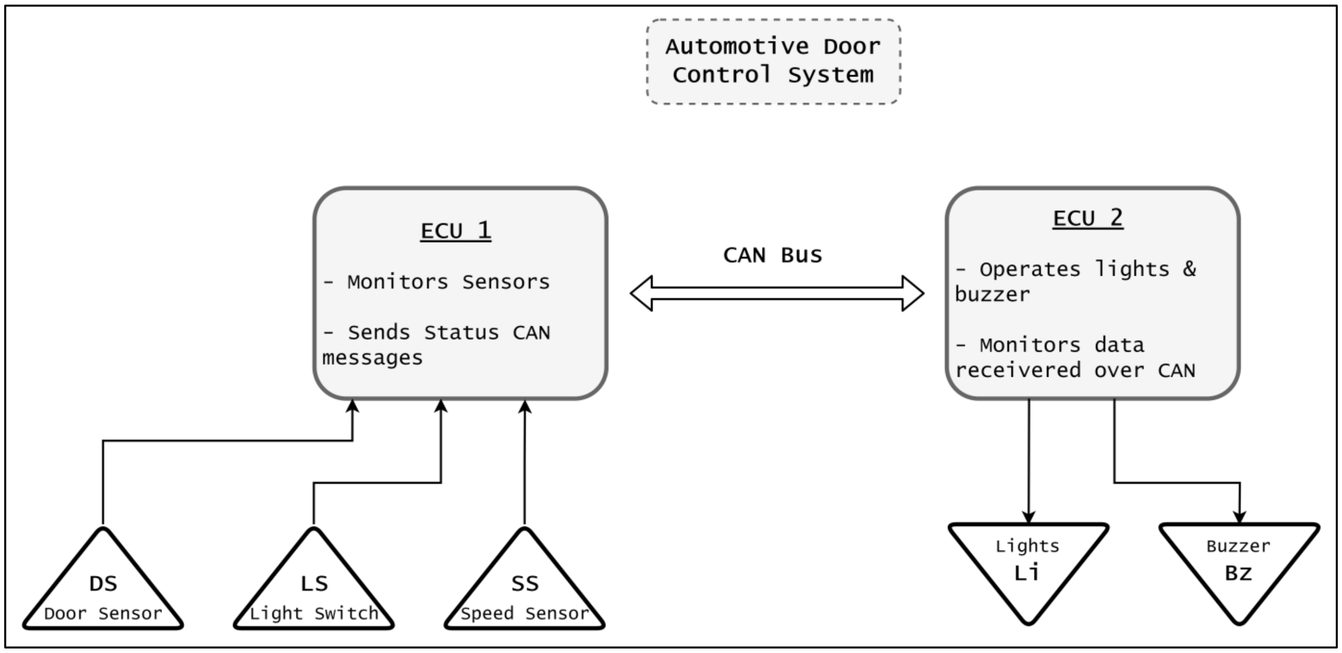
Automotive Door Control System

Static Design

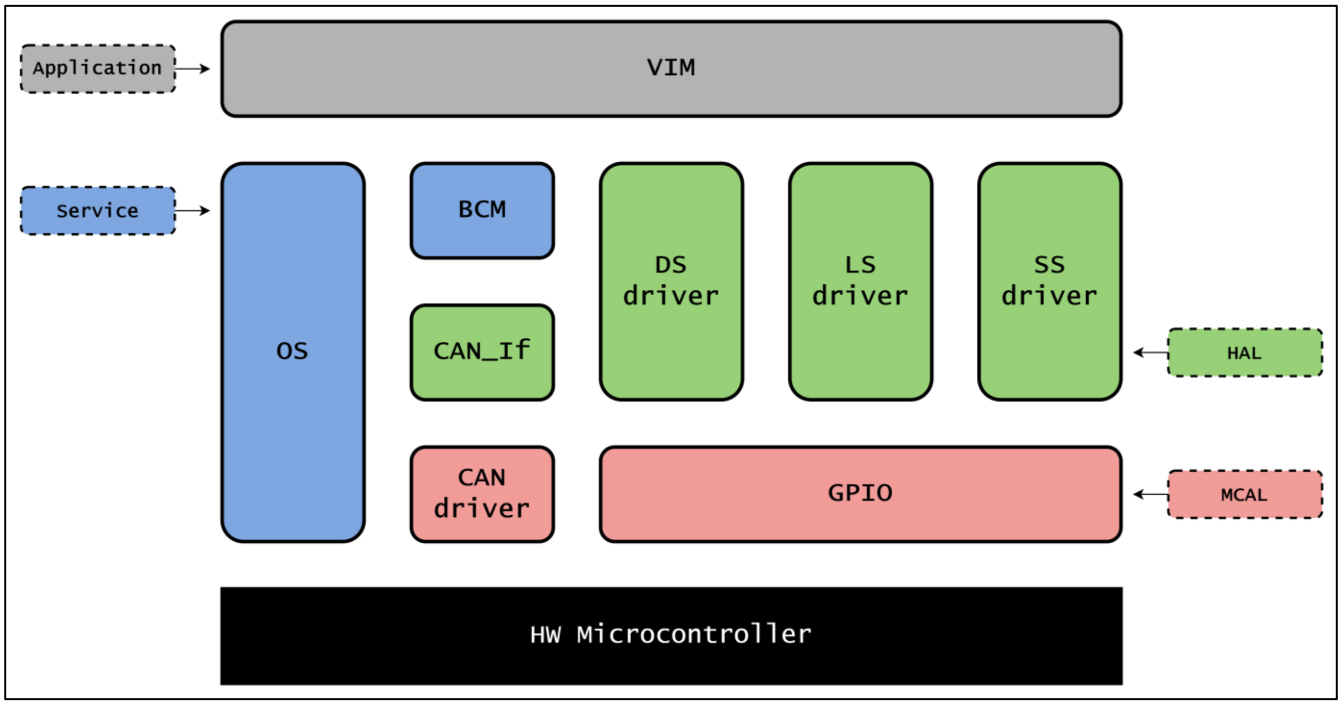
Owner: Mohamed Hossam , Email: mohamed.hossam.1183@gmail.com

1 System Schematic

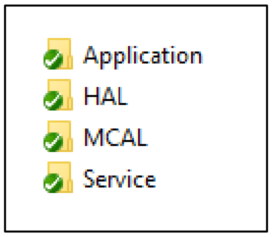

Automotive Door Control System
ECU 1
- Monitors Sensors
- Sends Status CAN messages
CAN Bus
ECU 2
- Operates lights & buzzer
- Monitors data receivered over CAN
DS
Door Sensor
LS
Light Switch
SS
Speed Sensor
Lights
Li
Buzzer
Bz

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:

- 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.

2.4.2 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.

2.4.3 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.

2.4.4 DS_drv: Door Sensor Driver:
- void DS_Init(void): Initialize DS_drv module.

- void DS_Get_Data(uint32 * Data): Get door sensor status.

2.4.5 LS_drv: Light Sensor Driver:
- void LS_Init(void): Initialize LS_drv module.

- void LS_Get_Data(uint32 * Data): Get light sensor status.

2.4.6 SS_drv: Speed Sensor Driver:
- void SS_Init(void): Initialize SS_drv module.

- void SS_Get_Data(uint32 * Data): Get speed sensor status.

2.4.7 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.

2.4.8 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.

2.4.9 CAN_drv: CAN driver:

- void CAN_Init(void): Initialize CAN_drv module.

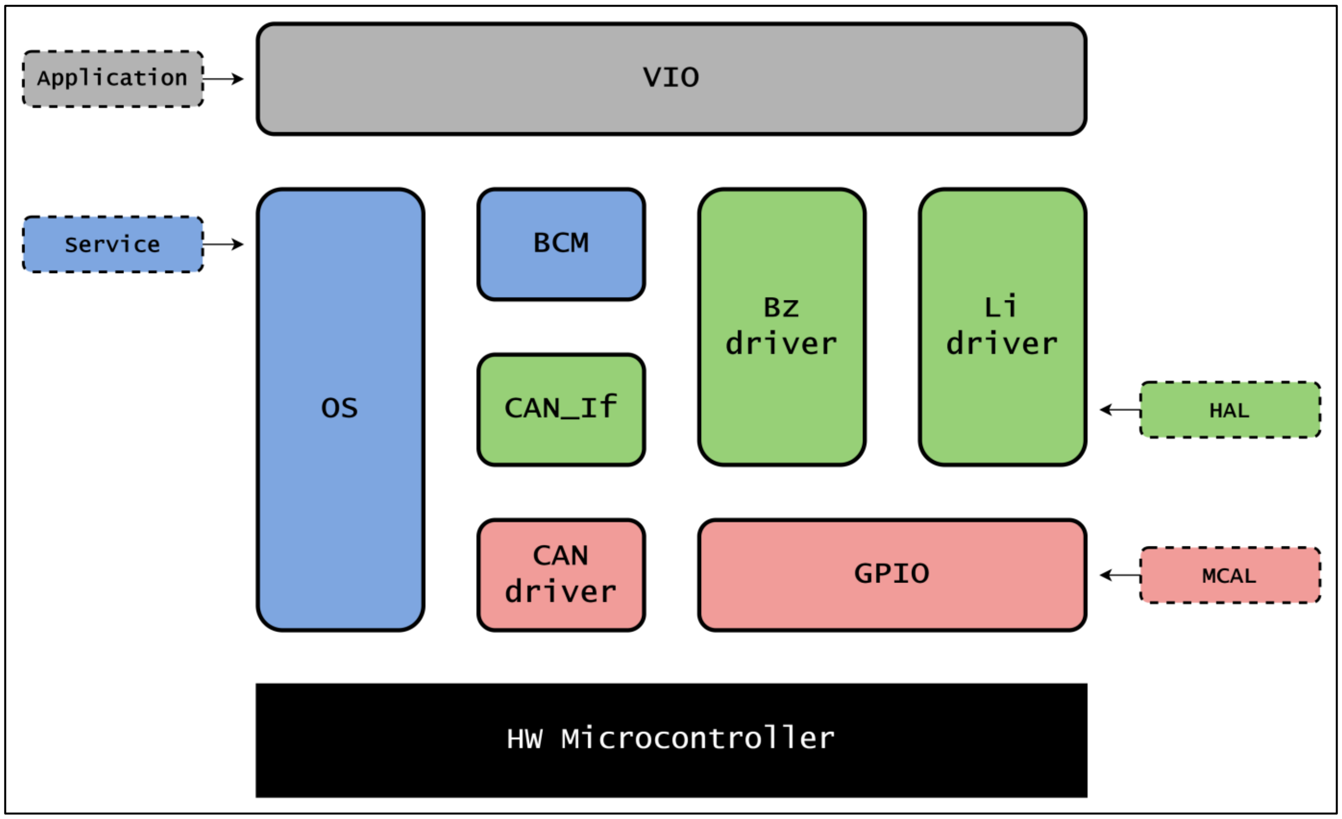
- void CAN_Set_Baudrate(uint32 Rate): Set CAN baud rate.

- Std_Result CAN_Write(uint8 * Data,
uint8 BuffSize): Write frame data to registers.

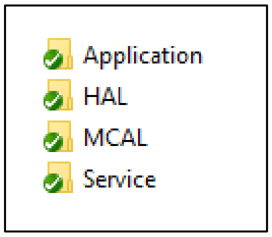
- Std_Result CAN_Read(uint8 * Data,
uint8 BuffSize): Write frame data to registers.

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:

```
- 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.
```

3.4.2 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.
```

3.4.3 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.

3.4.4 Bz_drv: Buzzer driver:

- void Bz_Init(void): Initialize Bz_drv module.

- void Bz_Set_Data(uint32 * Data): Set Buzzer ON or OFF.

3.4.5 Li_drv: Lights driver:

- void Li_Init(void): Initialize Li_drv module.

- void Li_Set_Data(uint32 * Data): Set Lights ON or OFF.

3.4.6 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.

3.4.7 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.

3.4.8 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.