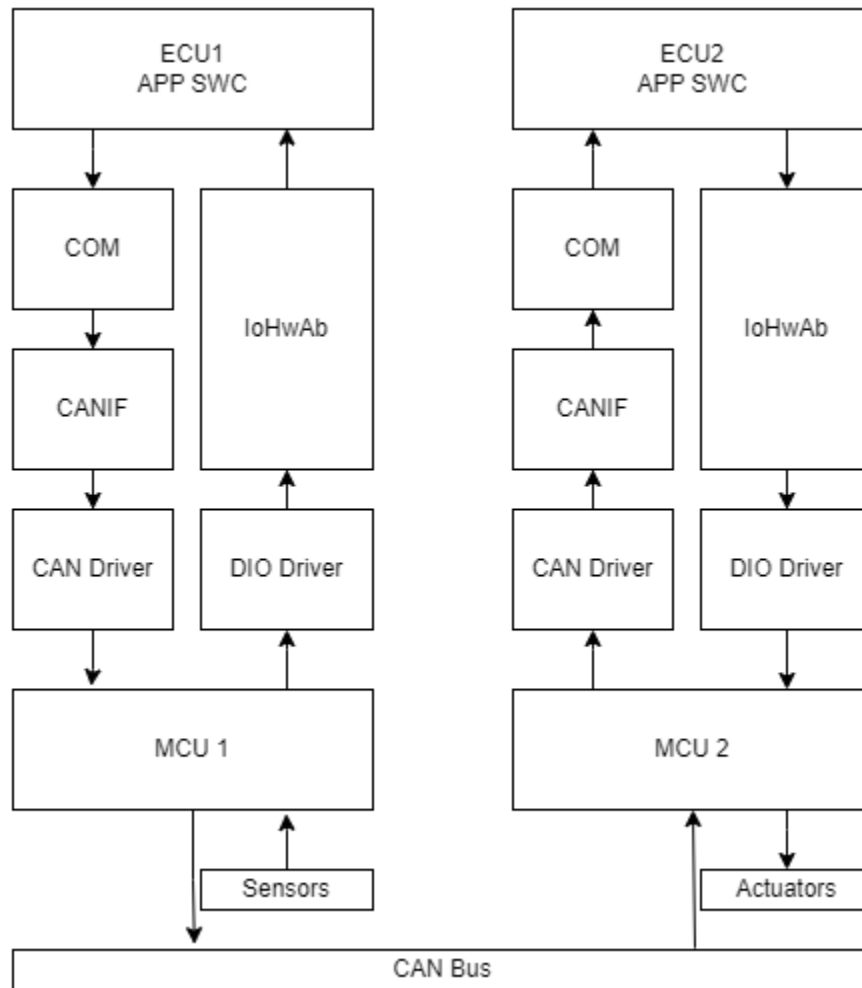


➤ **System Block Diagram**



1. Layered Architecture & ECU Components and Modules

1. Communication

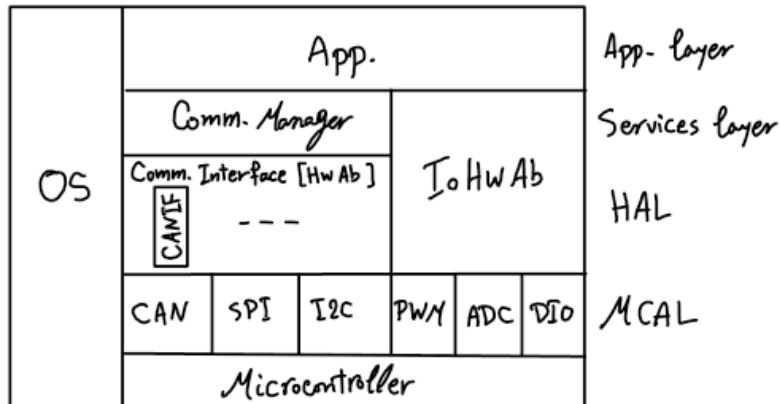
- The same layered architecture is used for both ECUs.
- The only drivers used in MCAL layer are DIO & CAN.
- If an external CAN peripheral is used with an SPI interface, then the SPI driver will be used instead of CAN.
- The CAN Interface (CANIF) in HAL or (ECU abstraction layer) is used to abstract which CAN peripheral is used, i.e. internal or external.

2. IO Hardware Abstraction

- Components in upper layers communicate with sensors and actuators via IO Hardware Abstraction.
- Provides access to MCAL drivers, e.g. DIO, PWM, ...etc.
- Provides DIO extension via SPI (for example).

3. OS

- Application layer methods, runnables, are called from within OS tasks.



2. Module APIs & description for the used typedefs

1. App Layer

ECU1 (3 Runnables): Lines 9:25

```
1  /*****
2  *                                     (4) App SWC
3  *****/
4
5  /* App -> IoHwAb -> DIO */
6  /* App -> Com -> CANIF -> CAN */
7
8  /* ECU 1 App Layer APIs */
9  void APP_DoorSensor(void)
10 {
11     IOHWAB_Read(IO_DoorSensor, &DoorState);
12     Com_Write(Com_DoorSensor, &DoorState);
13 }
14
15 void APP_SpeedSensor(void)
16 {
17     IoHwAb_Read(IO_SpeedSensor, &SpeedState);
18     Com_Write(Com_SpeedSensor, &SpeedState);
19 }
20
21 void APP_LightSwitch(void)
22 {
23     IoHwAb_Read(IO_LightSwitch, &LightSwitchState);
24     Com_Write(Com_LightSwitch, &LightSwitchState);
25 }
```

ECU2 (1 Runnable) : Lines 28:39

```
27 /* ECU 2 App Layer APIs */
28 void App_ReadSensor(void)
29 {
30     /* Read Sensor Data via CAN Bus */
31     /* Return status: indicates valid/invalid received signal. */
32     Com_Read(Com_SpeedSensor, &SpeedState);
33     Com_Read(Com_DoorSensor, &DoorState);
34     Com_Read(Com_LightSwitch, &LightSwitchState);
35
36     /* Check State Machine Logic of Actuators */
37     /* Control the Actuators via IoHwAb Component */
38     StateMachineLogic();
39 }
```

2. Comm Manager (BCM) and IoHwAb Component

ECU1: utilizes Com_Write(), IoHwAb_Read()

ECU1: utilizes Com_Read(), IoHwAb_Write()

```
41  /*****
42  *                                     (3) COM/ (2-3) IOHWAB
43  *****/
44
45  /* Comm Manager */
46
47  /* DoorState / LightSwitchState / SpeedState / 1-Byte? */
48  void Com_Write(Signal_t signal_id, void* data_p);
49  void Com_Read(Signal_t signal_id, void* data_p);
50
51  /* IoHwAb Component */
52  void IoHwAb_Read(Device_t dev_id, void* data_p);
53  void IoHwAb_Write(Device_t dev_id, void* data_p);
54
55  /*
56  1. Comm Manager typedefs
57  I. Signal_t: represents signal id, typically a uint8 data type.
58  values: Com_DoorSensor, Com_SpeedSensor, Com_LightSwitch
59
60  2. IoHwAb typedefs:
61  I. Device_t: represents device id, typically a uint8 data type.
62  values: IO_DoorSensor, IO_SpeedSensor, IO_LightSwitch
63  */
64
```

3. DIO Driver (MCAL)

ECU1: utilizes Dio_Init(), Dio_ReadChannel()

ECU1: utilizes Dio_Init(), Dio_WriteChannel()

```
65  /*****
66  *                                     (2) DIO
67  *****/
68  void Dio_Init(const DioConfig_t* DioConfig_p);
69
70  DioLevel_t Dio_ReadChannel(DioChannel_t ChannelId);
71
72  void Dio_WriteChannel(DioChannel_t ChannelId, DioLevel_t Level);
73
74
75  /*
76  1. DIO Driver typedefs
77  I. DioConfig_t: structure to hold Dio Config;
78  Channel ID, direction and initial value for output channels.
79  II. DioChannel_t: represents channel ID, typically
80  Data Type: uint8 data type.
81  III. DioLevel_t: represents DIO channel level (0 or 1).
82  Data Type: uint8 data type.
83  */
```

4. CANIF (CAN Interface Component in HAL)

ECU1: utilizes CanIf_Init(), CanIf_Transmit()

ECU1: utilizes CanIf_Init(), CanIf_ReadRxData()

```
85  /******
86  *                                     (2) CANIF
87  *****/
88  void CanIf_Init(const CanIfConfig_t* CanIfConfig_p);
89
90  void CanIf_Transmit(PduId_t PduId, const PduInfo_t* CanIfTxInfo_p);
91  void CanIf_ReadRxData(PduId_t PduId, PduInfo_t* CanIfRxInfo_p);
92
93  /*
94  1. CANIF typedefs
95     I. CanIfConfig_t: structure to hold the configuration of CAN.
96     II. PduId_t: specifies CAN Msg. ID, and implicitly CAN Driver Id.
97         Data Type: uint8, uint16
98
99     III. PduInfo_t: structure variable that holds a pointer to data,
100        and an integer variable representing data length.
101     IV. PduLength_t: to hold data length of a CAN frame (Max. 8)
102        Data Type: uint8
103
104  struct PduInfo_t{
105      uint8* Data_p;
106      PduLength_t DataLength;
107  }
108  */
```

5. CAN Driver (MCAL)

ECU1: utilizes Can_Init(), Can_Write()

ECU1: utilizes Can_Init(), Can_ReadRxBuffer()

```
110  /******
111  *                                     (1) CAN Driver
112  *****/
113  void Can_Init(const CanConfig_t* CanConfig_p);
114
115  void Can_Write(CanHwHandle_t CanHwHandle, const CanPduInfo_t* CanPduInfo_p);
116
117  void Can_ReadRxBuffer(CanHwHandle_t CanHwHandle, const CanPduInfo_t* CanPduInfo_p);
118
119
120  /*
121  1. CAN Driver typedefs
122     I. CanConfig_t: structure to hold the configuration of CAN Peripheral.
123     II. CanPduInfo_t: structure to specify CAN message ID, pointer to data,
124         and data length.
125     III. CanHwHandle_t: to represent CAN hardware object handle
126         (handle to PDU buffer inside CAN RAM, i.e. CAN Hardware Unit).
127         Data Type: uint8 or uint16 (depending on #CAN hardware units).
128     IV. CanId_t: to represent Message ID.
129         Data Type: uint32
130     V. PduLength_t: to hold data length of a CAN message (Max. 8)
131         Data Type: uint8
132
133     struct CanPduInfo_t{
134         // Data Length. (Max: 8)
135         PduLength_t DataLength;
136         // Can Msg. Identifier
137         CanId_t CanId;
138         // Data Buffer
139         const uint8* data_p;
140     };
141
142     Note:
143     1. Hardware Filtering:
144         Reception: enable Hardware Filtering of messages based on Message Identifier (ID).
145         Specify (Single ID, IDs from a list, or IDs from specified range).
146  */
```

3. Folder Structure

