1. Layered Architecture & ECU Components and Modules

1. Communication

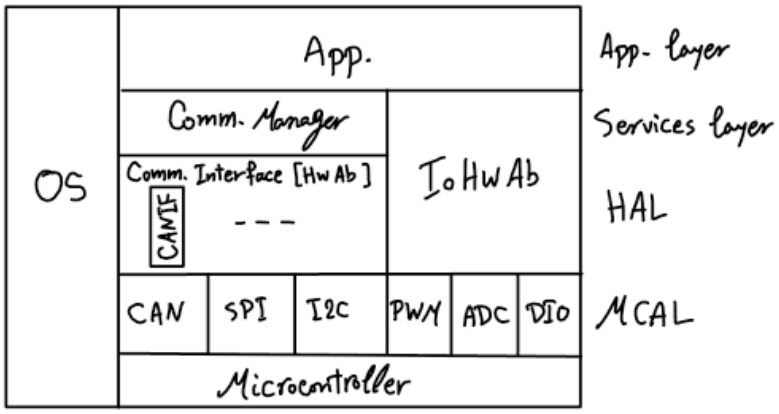
- **The same layered architecture is used for both ECUs.**
- **The only drivers used in MCAL layer are DIO & CAN.**
- **If an external CAN peripheral is used with an SPI interface, then the SPI driver will be used instead of CAN.**
- **The CAN Interface (CANIF) in HAL or (ECU abstraction layer) is used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

```c
/*******************************************************************************
 *                                    (4) App SWC
 *******************************************************************************/

/* App -> IoHwAb -> DIO */
/* App -> Com -> CANIF -> CAN */

/* ECU 1 App Layer APIs */
void APP_DoorSensor(void)
{
    IOHWAB_Read(IO_DoorSensor, &DoorState);
    Com_Write(Com_DoorSensor, &DoorState);
}

void APP_SpeedSensor(void)
{
    IoHwAb_Read(IO_SpeedSensor, &SpeedState);
    Com_Write(Com_SpeedSensor, &SpeedState);
}

void APP_LightSwitch(void)
{
    IoHwAb_Read(IO_LightSwitch, &LightSwitchState);
    Com_Write(Com_LightSwitch, &LightSwitchState);
}

/* ECU 2 App Layer APIs */
void App_ReadSensor(void)
{
    /* Read Sensor Data via CAN Bus */
    /* Return status: indicates valid/invalid received signal. */
    Com_Read(Com_SpeedSensor, &SpeedState);
    Com_Read(Com_DoorSensor, &DoorState);
    Com_Read(Com_LightSwitch, &LightSwitchState);

    /* Check State Machine Logic of Actuators */
    /* Control the Actuators via IoHwAb Component */
    StateMachineLogic();
}
```

2. Comm Manager (BCM) and IoHwAb Component

```
/**********************************************************************************
 *                                    (3) COM/ (2-3) IOHWAB
 **********************************************************************************/

/* Comm Manager */

/* DoorState / LightSwitchState / SpeedState / 1-Byte? */
void Com_Write(Signal_t signal_id, void* data_p);
void Com_Read(Signal_t signal_id, void* data_p);

/* IoHwAb Component */
void IoHwAb_Read(Device_t dev_id, void* data_p);
void IoHwAb_Write(Device_t dev_id, void* data_p);

/*
1. Comm Manager typedefs
   I. Signal_t: represents signal id, typically a uint8 data type.
      values: Com_DoorSensor, Com_SpeedSensor, Com_LightSwitch

2. IoHwAb typedefs:
   I. Device_t: represents device id, typically a uint8 data type.
      values: IO_DoorSensor, IO_SpeedSensor, IO_LightSwitch
*/

```

3. DIO Driver (MCAL)

```
/**********************************************************************************
 *                                    (2) DIO
 **********************************************************************************/
void Dio_Init(const DioConfig_t* DioConfig_p);

DioLevel_t Dio_ReadChannel(DioChannel_t ChannelId);

void Dio_WriteChannel(DioChannel_t ChannelId, DioLevel_t Level);


/*
1. DIO Driver typedefs
   I. DioConfig_t: structure to hold Dio Config;
      Channel ID, direction and initial value for output channels.
  II. DioChannel_t: represents channel ID, typically
      Data Type: uint8 data type.
 III. DioLevel_t: represents DIO channel level (0 or 1).
      Data Type: uint8 data type.
*/
```

4. CANIF (CAN Interface Component in HAL)

```
/*******************************************************************************
*                                    (2) CANIF
*******************************************************************************/
void CanIf_Init(const CanIfConfig_t* CanIfConfig_p);

void CanIf_Transmit(PduId_t PduId, const PduInfo_t* CanIfTxInfo_p);
void CanIf_ReadRxData(PduId_t PduId, PduInfo_t* CanIfRxInfo_p);

/*
1. CANIF typedefs
   I. CanIfConfig_t: structure to hold the configuration of CAN.
  II. PduId_t: specifies CAN Msg. ID, and implicitly CAN Driver Id.
      Data Type: uint8, uint16

 III. PduInfo_t: structure variable that holds a pointer to data,
      and an integer variable representing data length.
  IV. PduLength_t: to hold data length of a CAN frame (Max. 8)
      Data Type: uint8

struct PduInfo_t{
   uint8* Data_p;
   PduLength_t DataLength;
}
*/
```

5. CAN Driver (MCAL)

```
/*******************************************************************************
 *                          (1) CAN Driver
 *******************************************************************************/
void Can_Init(const CanConfig_t* CanConfig_p);

void Can_Write(CanHwHandle_t CanHwHandle, const CanPduInfo_t* CanPduInfo_p);

void Can_ReadRxBuffer(CanHwHandle_t CanHwHandle, const CanPduInfo_t* CanPduInfo_p);


/*
1. CAN Driver typedefs
   I. CanConfig_t: structure to hold the configuration of CAN Peripheral.
  II. CanPduInfo_t: structure to specify CAN message ID, pointer to data,
      and data length.
 III. CanHwHandle_t: to represent CAN hardware object handle
      (handle to PDU buffer inside CAN RAM, i.e. CAN Hardware Unit).
      Data Type: uint8 or uint16 (depending on #CAN hardware units).
  IV. CanId_t: to represent Message ID.
      Data Type: uint32
   V. PduLength_t: to hold data length of a CAN message (Max. 8)
      Data Type: uint8

struct CanPduInfo_t{
   // Data Length. (Max: 8)
   PduLength_t DataLength;
   // Can Msg. Identifier
   CanId_t CanId;
   // Data Buffer
   const uint8* data_p;
};

Note:
1. Hardware Filtering:
   Reception: enable Hardware Filtering of messages based on Message Identifier (ID).
   Specify (Single ID, IDs from a list, or IDs from specified range).
*/
```

3. Folder Structure