

Automotive Door Control System Design

Static Design

Nora El-Hennawy | EGYFWD | 4-10-22

# System Block Diagram

## 

D :Door Sensor B: Buzzer

S: Speed Sensor RL: Right Light

L: Light Switch LL: Left Light

## Assumptions:

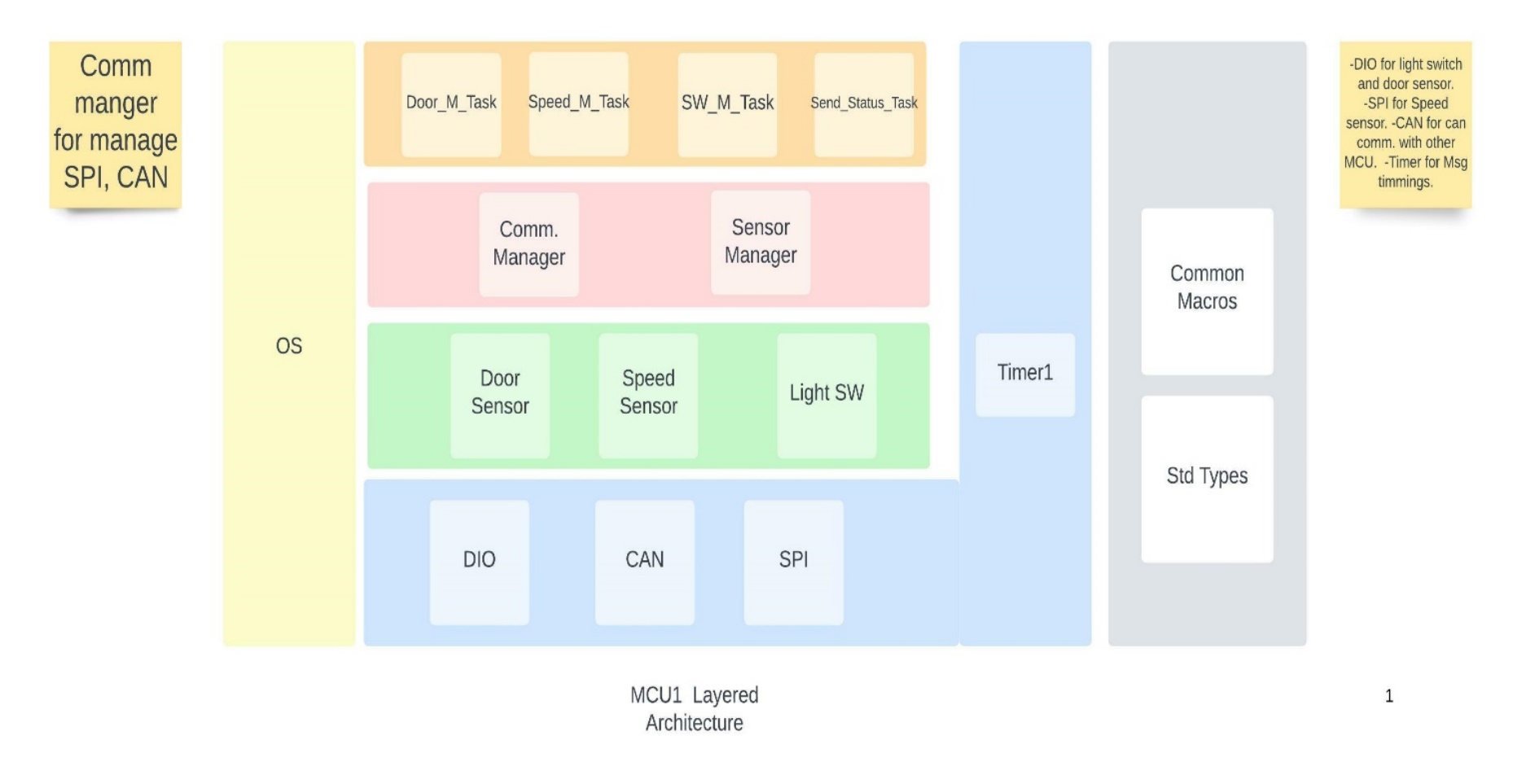
ECU1 has an input capture unit that will be used to take readings from speed sensors and send it on SPI??

DIO Ports on ECU are 16 Pin/Port.

NOTE: If OS used provide Delay function no need for Timer driver.

# ECU1

## Layered Architecture Diagram



## ecu1 Compponents and modules

DIO

CAN

SPI

Timer

SPEED\_Sensor

Light\_SW

Door\_Sensor

Communication Manager:

Sensor Manager:

Tasks:

Status Queue

## ECU1 API Function Description

### DIO:

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: void API\_DIO\_Init(void) | |  | |
| Arguments | Inputs: None | A1: | Type: |
| Description: | |
| A2: | Type |
| Description: | |
| Return | R: | Type: | |
| Description: |  | |
| Description | Initialize DIO driver with user configuration | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: DIOPinVal API\_DIO\_ReadPin(DIOPinID PinNum, DIOPortID PortID) | | | |
| Arguments | Inputs: PinNum ,PortID | A1: PinNum | Type: DIOPinID |
| Description: Pin number to read. | |
| A2:PortId | Type: DIOPortID |
| Description: ID for Port that contains PinNum | |
| Return | R: PinVal | Type: DIOPinVal | |
| Description: Pin value for PinNum on PortID | | |
| Description | Read Value fro PinNum on PortID | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: DIOPortVal API\_DIO\_ReadPort(DIOPortID PortID) | | | |
| Arguments | Inputs: PortID | A1: PortId | Type: DIOPortID |
| Description: ID for Port to read from. | |
| A2: | Type: |
| Description: | |
| Return | R: PortVal | Type: DIOPortVal | |
| Description: current reading of Port value for PortID | | |
| Description | Read Value for PortId | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: void API\_DIO\_WritePort(DIOPortID PortID, DIOPortVal PortVal) | | | |
| Arguments | Inputs: PortID , PortVal | A1: PortId | Type: DIOPortID |
| Description: ID for Port to write on. | |
| A2:PortVal | Type: DIOPortVal |
| Description: Value to write on Port PortID. | |
| Return | R: None | Type: | |
| Description: | | |
| Description | Writes Value :PortVal on Port: PortId | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: void API\_DIO\_WritePin(DIOPortID PortID, DIOPinID PinNum, DIOPinVal PinVal) | | | |
| Arguments | Inputs: PortID ,PinNum, PinVal | A1: PortId | Type: DIOPortID |
| Description: ID for Port to write on. | |
| A2:PinNum | Type: DIOPinID |
| Description: PinNum to write on Port: PortID. | |
| A3:PinVal | Type: DIOPinVal |
| Description: Value to write on Pin :PinNum on Port: PortID. | |
| Return | R: None | Type: | |
| Description: | | |
| Description | Writes Value :PinVal on Pin:PinNum on Port: PortId | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: void API\_DIO\_TogglePin(DIOPinID PinNum, DIOPortID PortID) | | | |
| Arguments | Inputs: PinNum ,PortID | A1: PinNum | Type: DIOPinID |
| Description: Pin number to toggle. | |
| A2:PortId | Type: DIOPortID |
| Description: ID for Port that contains PinNum | |
| Return | R: None | Type: | |
| Description: | | |
| Description | Toggle PinNum on PortID | | |

## Type Defines:

* DIOPinVal :

**DataType**: u8 **Description**: DIO Pin value ( 0/1)

* DIOPortVal :

**DataType**: u8 **Description**: DIO Port value ( 0x00🡪0xff)

* DIOPinID :

**DataType**: u8 **Description**: DIO Pin number ( 0 🡪 8)

* DIOPortID :

**DataType**: u16 **Description**: DIO Port Id ( PortA,PortB….)

## CAN

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: void API\_CAN\_Init(void) | |  | |
| Arguments | Inputs: None | A1: | Type: |
| Description: | |
| A2: | Type |
| Description: | |
| Return | R: | Type: | |
| Description: |  | |
| Description | Initialize CAN driver with user configuration | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: void API\_CAN\_Write(CAN\_Msg Msg) | |  | |
| Arguments | Inputs: Msg | A1:Msg | Type: CANMsg |
| Description: Message to send on Can Bus | |
| A2: | Type |
| Description: | |
| Return | R: | Type: | |
| Description: |  | |
| Description | Send message “Msg” on Can Bus | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: u8 API\_CAN\_Read(CAN\_Msg \* Msg) | |  | |
| Arguments | Inputs: Msg | A1:Msg | Type: |
| Description: pointer to can message structure to return message read from can bus in it. | |
| A2: | Type |
| Description: | |
| Return | R: NewMsg | Type:Bool | |
| Description: return True if new message was received from can bus else return False. | | |
| Description | Fill” Msg” with new message received from can bus else return False. | | |

## Type Defines:

CAN\_Msg

Type: Structure Description: Structure containing CAN message (vary according to used Can protocol)

## SPI

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: void API\_SPI\_Init(void) | |  | |
| Arguments | Inputs: None | A1: | Type: |
| Description: | |
| A2: | Type |
| Description: | |
| Return | R: | Type: | |
| Description: |  | |
| Description | Initialize SPI driver with user configuration | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: void API\_SPI\_Write(SPI\_Msg Msg) | |  | |
| Arguments | Inputs: Msg | A1:Msg | Type: SPI\_Msg |
| Description: Message to send on SPI | |
| A2: | Type |
| Description: | |
| Return | R: | Type: | |
| Description: |  | |
| Description | Send message “Msg” on SPI | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: u8 API\_SPI\_Read(SPI\_Msg \* Msg) | |  | |
| Arguments | Inputs: Msg | A1:Msg | Type: |
| Description: pointer to SPI message structure to return message read from SPI in it. | |
| A2: | Type |
| Description: | |
| Return | R: NewMsg | Type:Bool | |
| Description: return True if new message was received from SPI bus else return False. | | |
| Description | Check if new message was received from SPI Fill” Msg” and return True else return False. | | |

## Type Defines:

SPI\_Msg

Type: Structure Description: Structure containing SPI message.

## Timer

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: void API\_Timer\_Init(void) | |  | |
| Arguments | Inputs: None | A1: | Type: |
| Description: | |
| A2: | Type |
| Description: | |
| Return | R: | Type: | |
| Description: |  | |
| Description | Initialize Timer driver with user configuration | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: void API\_SetTimerCallBack(FncPtr\* CallBackFnc) | | | |
| Arguments | Inputs:  CallBackFnc | A1:CallBackFnc | Type: pointer to function |
| Description: pointer to call back function to call in Timer ISR | |
| A2: | Type |
| Description: | |
| Return | R: | Type: | |
| Description: |  | |
| Description | Assign call back function to be called in Timer ISR | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: void API\_DelayMs(u16 Delay) | |  | |
| Arguments | Inputs: Delay | A1: Delay | Type: u16 |
| Description: Value of delay to wait in milliseconds | |
| A2: | Type |
| Description: | |
| Return | R: | Type: | |
| Description: |  | |
| Description | Wait for (Delay) ms | | |

NOTE: If OS used provide Delay function no need for Timer driver.

## SPEED\_Sensor

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: S\_SensorState API\_GetSpeedSensorState(SensorID SensorNum) | | | |
| Arguments | Inputs:  SensorNum | A1: SensorNum | Type: SensorID |
| Description: speed sensor number. | |
| A2: | Type |
| Description: | |
| Return | R: SensorState | Type:S\_SensorState | |
| Description: Speed Sensor State | | |
| Description | Return Speed Sensor State for given sensor number | | |

## Type Defines:

* SensorID

Type: enum Description: unique identifier for each sensor

* S\_SensorState

Type: enum Description: unique identifier for speed sensor states(CAR\_IS\_MOV,CAR\_STOPPED)

## Light\_SW

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: L\_SW\_State API\_Get\_L\_SWState(L\_SW\_ID L\_SW\_Num) | | | |
| Arguments | Inputs:  L\_SW\_Num | A1: L\_SW\_Num | Type: L\_SW\_ID |
| Description: Light SW number. | |
| A2: | Type |
| Description: | |
| Return | R: SwState | Type: L\_SW\_State | |
| Description: Light switch State | | |
| Description | Return Light switch State for given light switch number | | |

## Type Defines:

* L\_SW\_Num

Type: enum Description: unique identifier for each light switch

* L\_SW\_State

Type: enum Description: unique identifier for light switch states(SW\_PRESSED,SW\_RELEASED)

## Door\_Sensor

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: D\_SensorState API\_GetDoorSensorState(SensorID SensorNum) | | | |
| Arguments | Inputs:  SensorNum | A1: SensorNum | Type: SensorID |
| Description: door sensor number. | |
| A2: | Type |
| Description: | |
| Return | R: SensorState | Type:S\_SensorState | |
| Description: Door Sensor State | | |
| Description | Return Door Sensor State for given sensor number | | |

## Type Defines:

* SensorID

Type: enum Description: unique identifier for each door sensor

* D\_SensorState

Type: enum Description: unique identifier for door sensor states(DOOR\_OPEN,DOOR\_CLOSED)

## sensor Manager

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: SensorState API\_ Get\_Sensor\_Status (SensType SensorType) | | | |
| Arguments | Inputs:  SensorType | A1: SensorNum | Type: SensorID |
| Description: speed sensor number. | |
| A2: SensorType | Type: SensType |
| Description: Sensor Type to get status for | |
| Return | R: SensorState | Type: void\* | |
| Description: Sensor State | | |
| Description | Return Sensor State for given sensor type and number  This is supervisor(generic)function to manage all system sensor with different types | | |

## Type Defines:

SensType

Type: enum Description: unique identifier for each sensor type in system(SPEEDSENSOR=0,DOORSENSOR=1,…)

* SensorState

Type: enum Description: unique identifier for sensor states(along with sensor type determine given sensor state)

## COMMUNICATION Manager

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: API\_Send\_Msg (COMM\_ID ComType , Msg\* Message) | | | |
| Arguments | Inputs:  ComType | A1: ComType | Type: COMM\_ID |
| Description: type of communication used to send Message | |
| A2:Message | Type void\* |
| Description: Message to be sent. | |
| Return | R: SendState | Type: SendState | |
| Description: Send Status (success/fail) | | |
| Description | Send message on given communication type and return sending status. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: API\_Rcev\_Msg (COMM\_ID ComType , Msg\* Message) | | | |
| Arguments | Inputs:  ComType | A1: ComType | Type: COMM\_ID |
| Description: type of communication used to receive Message | |
| A2:Message | Type void\* |
| Description: Message received. | |
| Return | R: R\_Status | Type: RcevState | |
| Description: Receive Status (new message/no message) | | |
| Description | Receive message on given communication type and return receive status. | | |

## Type Defines:

* COMM\_ID

Type: enum Description: unique identifier for each communication type(CAN=0,SPI=1,….)

* SendState

Type: enum Description: unique identifier for sending status(FAIL=0,SUCCESS=1)

## ECU1 Folder Structure

# ECU2

## Layered Architecture Diagram

## 

## ecu2 Compponents and modules

## ECU2 API Function Description

## DIO ,CAN ,COMM\_MANAGER

Refer to same modules in ECU1 [API Function description](#_ECU1_API_Function)

## BUZZER

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: BUZZER\_ON(BUZZER\_ID B\_Num) | | | |
| Arguments | Inputs:  B\_Num | A1: B\_Num | Type: BUZZER\_ID |
| Description: Buzzer number to turn ON. | |
| A2: | Type |
| Description: | |
| Return | R: | Type: | |
| Description: | | |
| Description | Turn Buzzer number :B\_Num ON. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: BUZZER\_OFF(BUZZER\_ID B\_Num) | | | |
| Arguments | Inputs:  B\_Num | A1: B\_Num | Type: BUZZER\_ID |
| Description: Buzzer number to turn OFF. | |
| A2: | Type |
| Description: | |
| Return | R: | Type: | |
| Description: | | |
| Description | Turn Buzzer number :B\_Num OFF. | | |

## Type Defines:

* BUZZER\_ID

Type: enum Description: unique identifier for each buzzer

## Light

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: Light\_ON(LIGHT\_ID L\_ID) | | | |
| Arguments | Inputs:  L\_Num | A1: L\_Num | Type:  LIGHT \_ID |
| Description: Light ID to turn ON. | |
| A2: | Type |
| Description: | |
| Return | R: | Type: | |
| Description: | | |
| Description | Turn Light with ID= L\_ID ON. | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: Light\_OFF(LIGHT\_ID L\_ID) | | | |
| Arguments | Inputs:  L\_Num | A1: L\_Num | Type:  LIGHT \_ID |
| Description: Light ID to turn OFF. | |
| A2: | Type |
| Description: | |
| Return | R: | Type: | |
| Description: | | |
| Description | Turn Light with ID= L\_ID OFF. | | |

## Type Defines:

* LIGHT\_ID

Type: enum Description: unique identifier for each buzzer(LEFT\_FRONT\_LIGHT=0,RIGHT\_FRONT\_LIGHT=1,…..)

## Actuator Manager

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: API\_ Set\_Actuator (ActType Actuator, Act\_ID Act\_num , ACTION\_ID Action) | | | |
| Arguments | Inputs:  Actuator , Act\_num, Action | A1: Actuator | Type: ActType |
| Description: Type of Actuator to set. | |
| A2: Act\_num | Type: Act\_ID |
| Description: Actuator number to SET | |
| A3:Action | Type:Action\_ID |
| Description:Action done on actuator(ON/OFF) | |
| Return | R: SensorState | Type: | |
| Description: Sensor State | | |
| Description | This is supervisor(generic)function to manage all system actuators with different types | | |

## Type Defines:

ActType

Type: enum Description: unique identifier for each actautor type in system(BUZZER=0,LIGHT=1,…)

Action\_ID

Type: enum Description: unique identifier for action to take place (OFF=0,ON=1)

## SysTEm CNTRL

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: UpdateSystemStatus(StatusMsg) | | | |
| Arguments | Inputs:  StatusMsg | A1: StatusMsg | Type:  STAT\_MSG |
| Description: Light ID to turn ON. | |
| A2: | Type |
| Description: | |
| Return | R: | Type: | |
| Description: | | |
| Description | Update system status(Door,LightSW,Car\_State) according to status message (extracted form status queue) | | |

|  |  |  |  |
| --- | --- | --- | --- |
| Function name: SystemCtrl() | | | |
| Arguments | Inputs: | A1: | Type: |
| Description: Light ID to turn ON. | |
| A2: | Type |
| Description: | |
| Return | R: | Type: | |
| Description: | | |
| Description | State machine that takes action according to current system status. (Door,LightSW,Car\_State) | | |

## Type Defines:

STAT\_MSG

Type: structure Description: structure for sensor status message received for ECU1.

## ECU2 Folder Structure