Icon

Description automatically generated

Nanodegree

Automotive door control design

Ahmed Elebaby

**Static design**

**Introduction:**

This PDF provides a comprehensive overview of the static design for an automotive door control system. It includes a detailed description of the layered architecture of the system, as well as the components and APIs of each layer. The purpose of this design is to illustrate the technical specifications and guidelines for implementing a reliable and efficient door control system in a vehicle. The information presented here will help stakeholders understand the system requirements and design considerations and provide a solid foundation for the implementation of the system.

The Layered Architectures For ECUS:

Apps

* The first Layered Architectures For ECUS:

State Tracer

Common

Modules

OS

&

Libs Serv

On system

Door Sensor Module

Light Switch Module

Light Speed Sensor Module

Module

Door Sensor Module

**CAN Bus Driver**

**INPUT Capture Driver**

**GPIO**

**Drive**

**TIMER**

**Driver**

Hardware AL

MCAL

* The second Layered Architectures For ECUS:

Apps

Common Module

Buzzer Module

Light Control

Data Handler

OS

&

Libs Serv

On system

Buzzer Control Module

Left Light Control Module

Module

Right Light Control Module

Module

Hardware AL

MCAL

**CAN Bus Driver**

**TIMER**

**Driver**

**GPIO**

**Drive**

Then, we outline the various components and modules within the ECU abstraction layer and its accompanying low-level drivers. Our approach begins with the lower layers in order to establish a solid foundation for the higher-level ECU abstraction layer and the application.

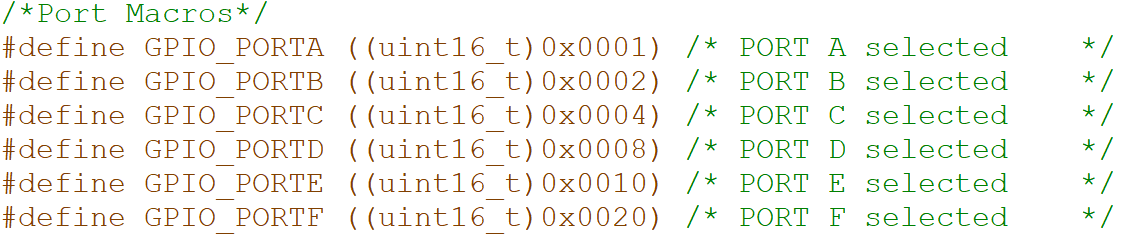
* **Low Layers:**
* Microcontroller Abstraction Layer:

**- GPIO (General purpose input-output):**

…describes the utilization of digital input/output in the ECU Abstraction Layer. The ECUAL layer will use it to communicate with the sensors and switches attached to the MCU. In order to control the operation of the GPIO module, full functional APIs must be provided for reading and writing data and controlling external interrupts on the pins.

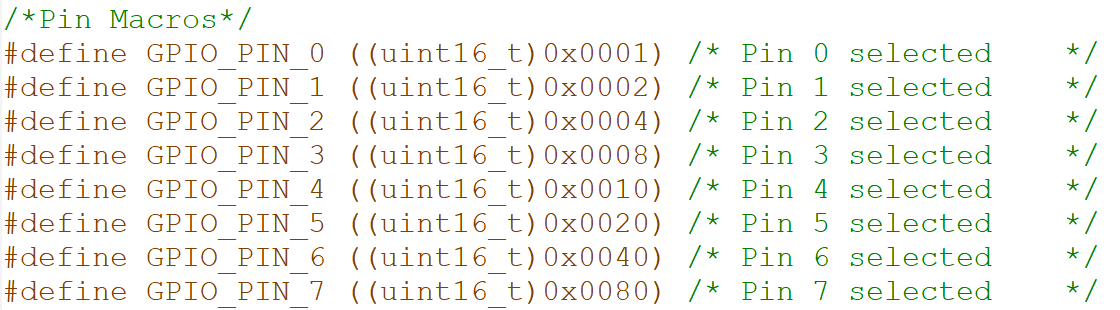
**API Type used for specifying the required PORT to control:**

- Macro to define GPIO ports of ECU



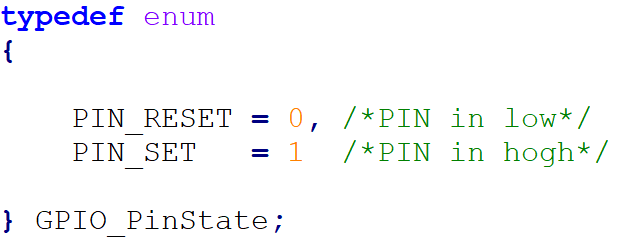
API Type for Port

**API Type used for specifying the required PIN to control:**



API Type for Pin I

**API Type enum used to read and control the pins state:**

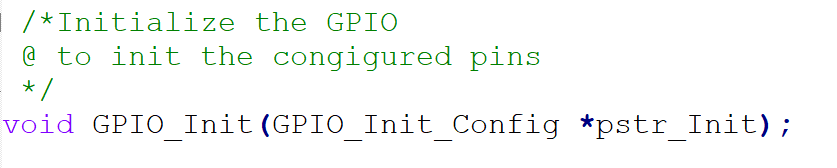
****

**API Type used to struct the configuration parameters and passing to the initializing API:**

**A picture containing text

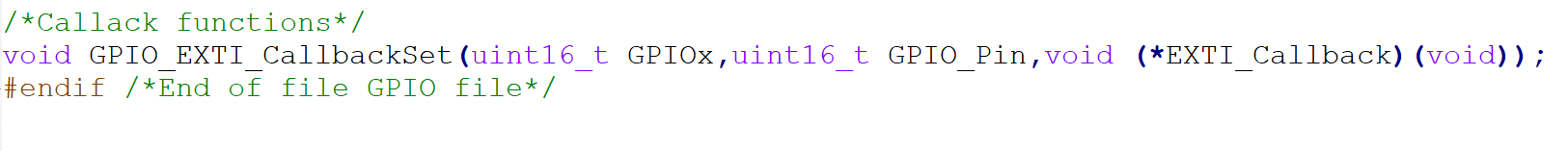
Description automatically generated**

**API functions to initialize the DIO module and control operations:**

****

**A picture containing text

Description automatically generated**

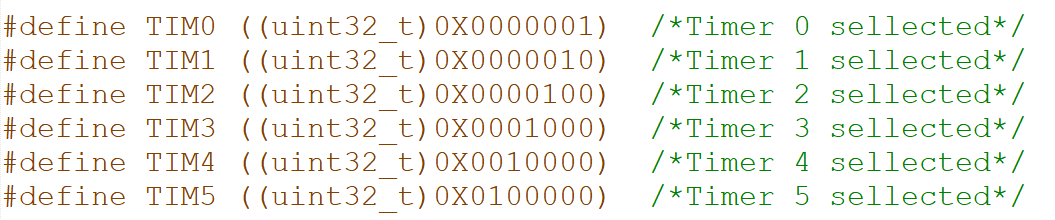
****

**- GPT (General purpose timer):**

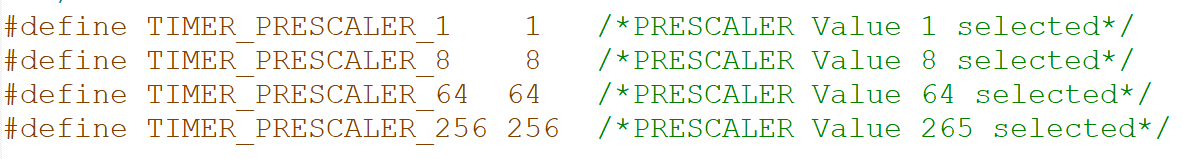
According to HRS, the target microcontroller is equipped to connect to multiple sensors that use the timer module for timing management and synchronization of the communication bus, as it regularly sends trace data over the CAN bus.

This driver must offer APIs that utilize the hardware timers in the MCU and create precise time-based event triggers for a specified number of times, and an API for provisioning.

**API Type used for initialization the channels:**

****

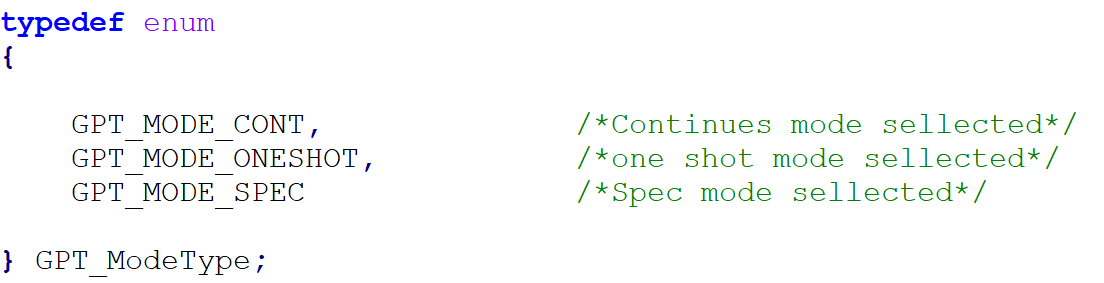
**API Type used to configure used perscaler:**

****

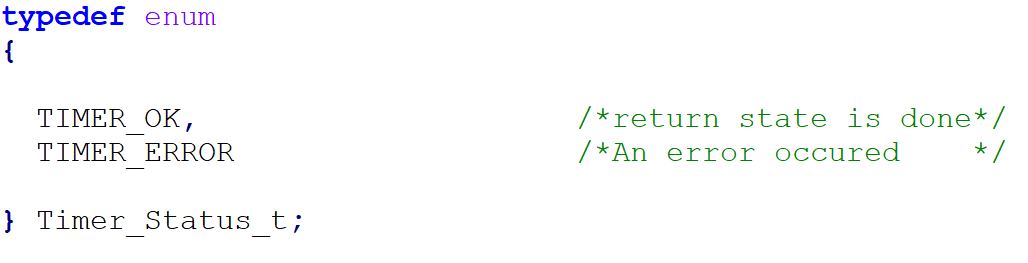
**API Type used to configure used clock type:**

****

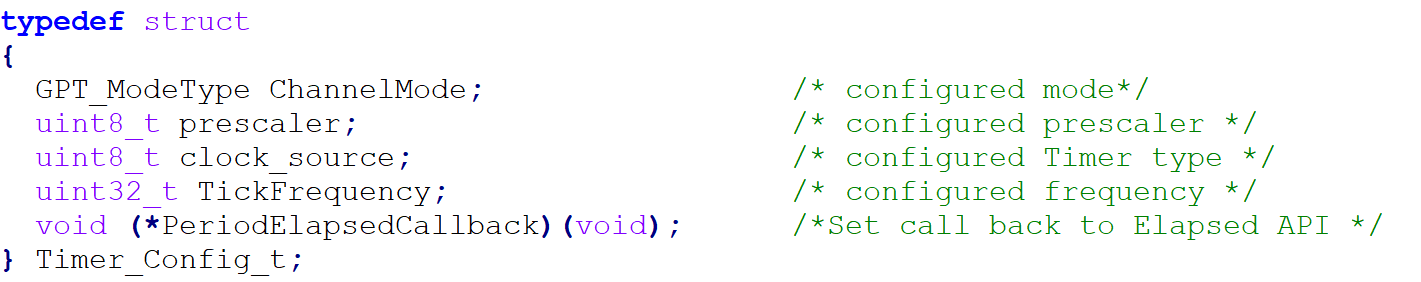
**API Type used to configure the modes:**

****

**API Type used to configure return state:**

****

**API Type used for struct the configuration parameters:**

****

**API functions used for initialization the driver and control operations:**

**Text

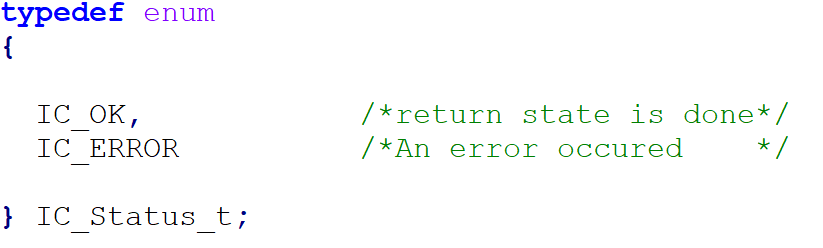
Description automatically generated**

- Input Capture Module:

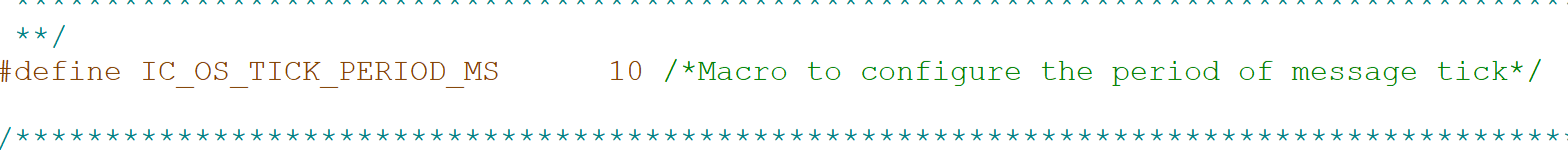
The IC Driver must offer APIs to accurately measure the duration between rising and falling edges of the signals received from the sensors. This driver utilizes a timer unit within the target system. The ECUAL layer relies on this driver to perform its component APIs, making precise implementation crucial for obtaining accurate sensor data. In this section.

We detail the types and functions of the module APIs.

**API Type used to configure return state:**

****

**API type used to identify periodicity:**

****

**API type to struct the configuration parameters of the module and pass it to the initialization API function:**

**Text

Description automatically generated**

**API functions to initialize the IC Module:**

**Text

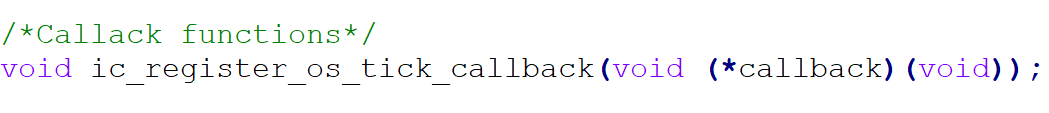
Description automatically generated with medium confidence**

**API functions to control the IC Module operation:**

**Text

Description automatically generated**

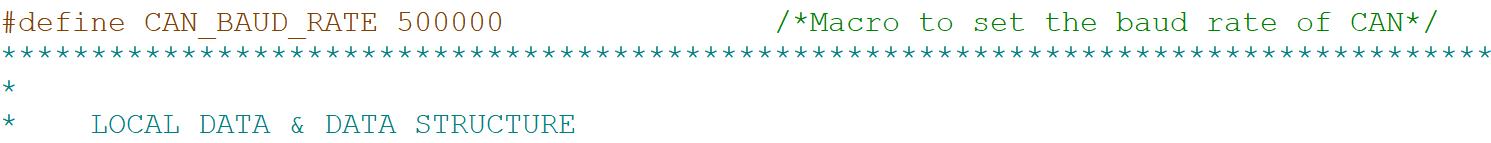
**API functions to set Call back function to IC module:**

****

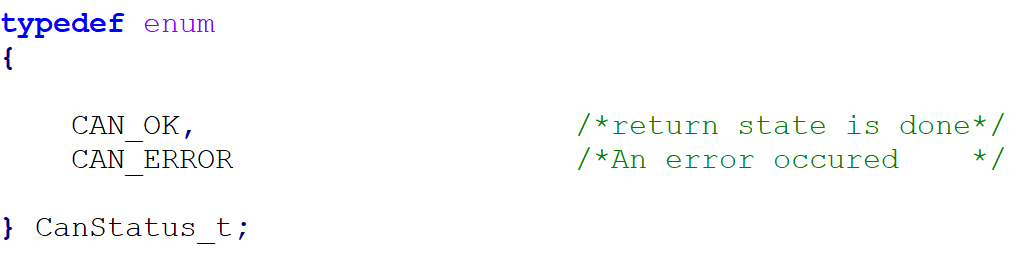
- CAN bus module:

The first ECU is tasked with transmitting collected data from sensors on a CAN bus periodically. In the higher levels of the architecture, the communication module is defined, and it relies on the APIs of the CAN driver to carry out its functions. Hence, the driver APIs have been defined to meet the necessary communication specifications.

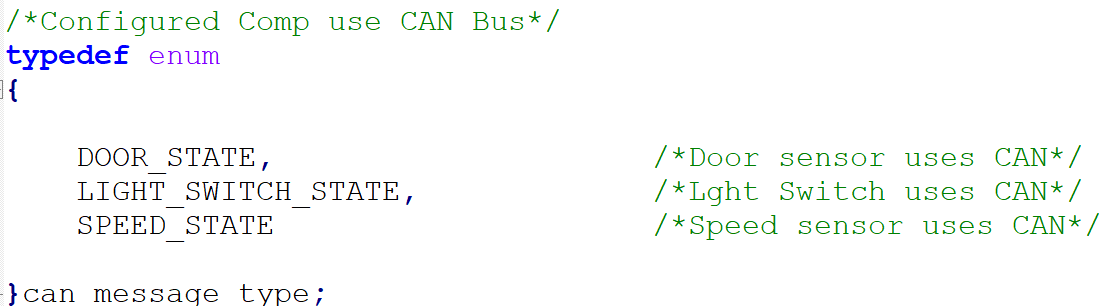
**API type used to identify baud rate for CAN bus:**



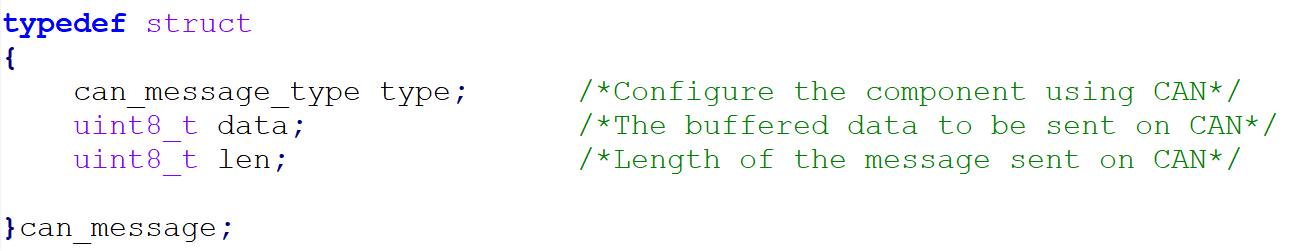
**API Type used to configure return state:**

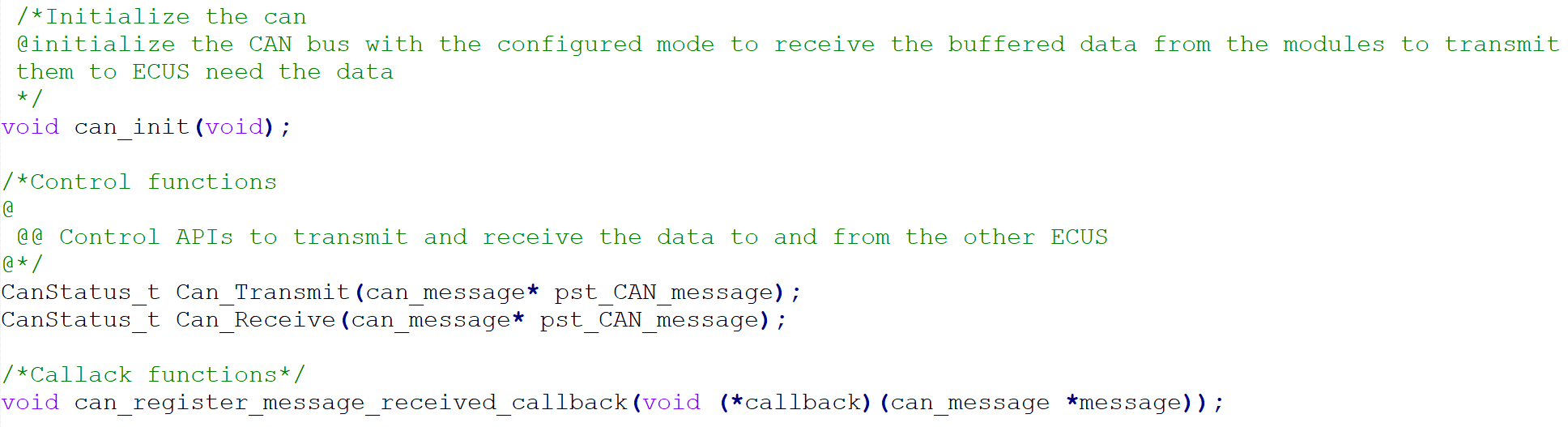
****

**API Type used to configure the module using CAN:**

****

**API Type used to struct the message sent by CAN:**

****

**API functions to allow initialization and control operations:**

**ECU Abstraction Layer**

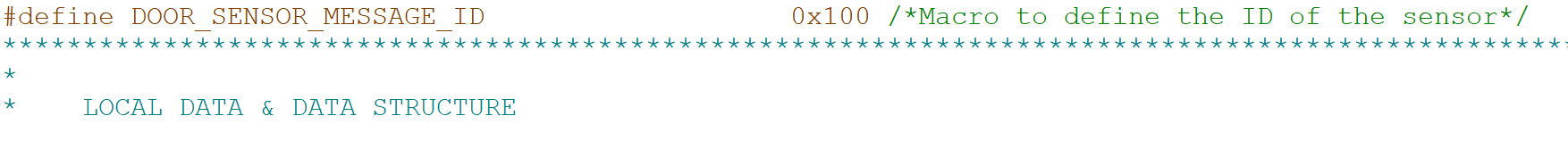
In this section, we outline the APIs of the ECU Abstraction Layer (ECUAL). The Application Layer will rely directly on these APIs. As the components in this layer vary between the two ECUs in the system, we will separately define each group of components. We begin with the components of ECU 1.

**ECU 1 Components:**

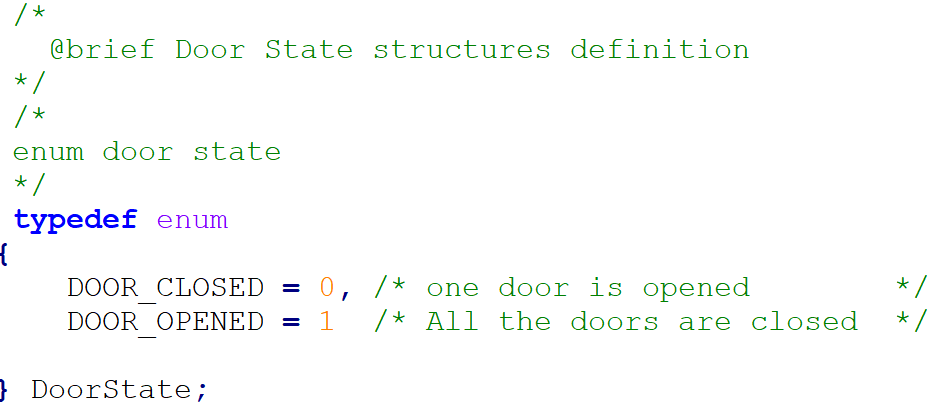
1. Door Sensor:

The sensor is expected to supply APIs for retrieving the state of the doors, as well as callbacks for any changes in the state.

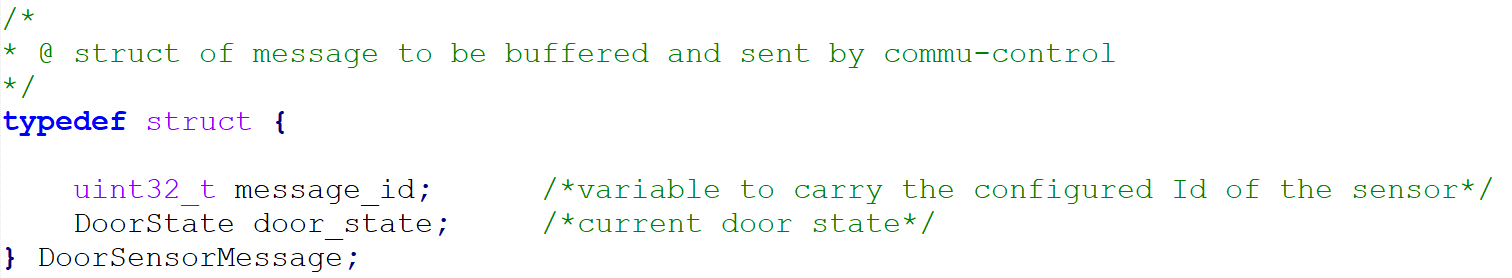
**API type to identify the ID for the message:**



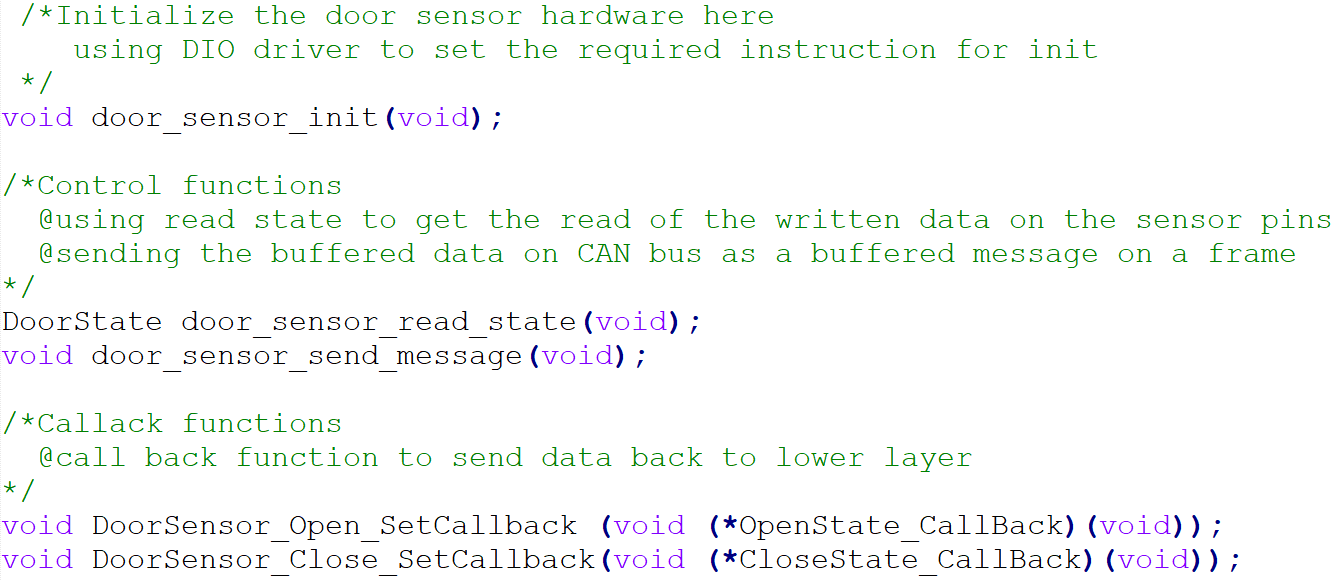
**API type for retrieving the door state:**



**API Type used to struct the message sent by Door sensor:**



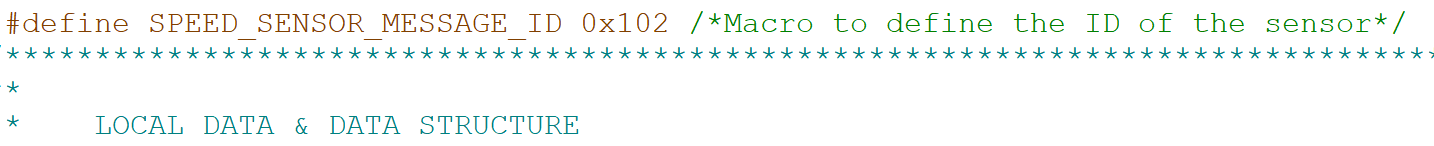
**API function to control operation and retrieve the state:**



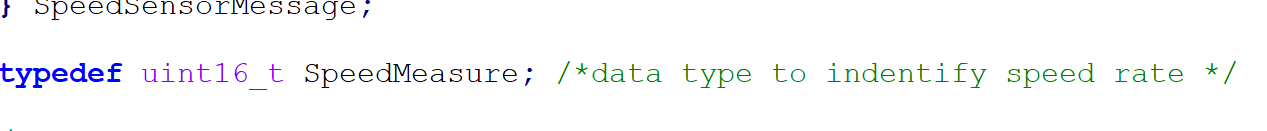
1. Speed Sensor:

This sensor is required to have API functions that can obtain measurements from the hardware sensor connected to the target. Additionally, the API should also be able to determine the state of the car's movement, with emphasis on the general state rather than a specific measurement for the speed.

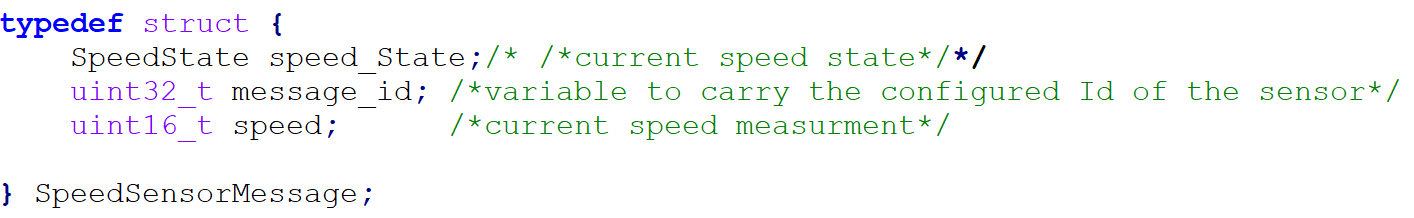
**API type to identify the ID for the message:**



**API to identify the type of speed:**



**API Type used to struct the message sent by Speed sensor:**



**API function to control operation and retrieve measurements and state:**

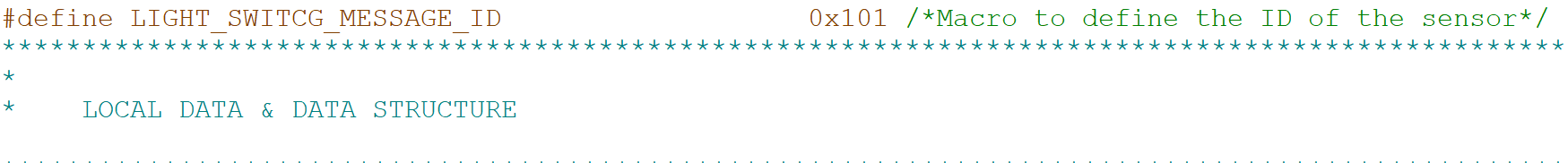
Text, letter

Description automatically generated

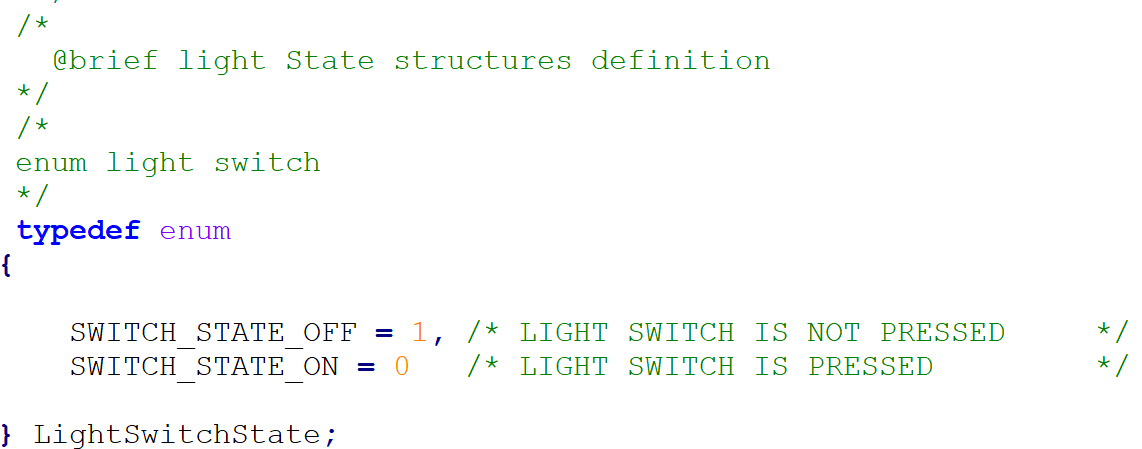
1. Light switch:

This component will be tasked with reading the state of the light switch input. It will provide an API that returns the state of the switch when it is called. To ensure accurate results, the implementation will take measures to account for any potential noise on the pins by repeatedly reading the state of the pins over a specified number of periods. These periods can be controlled through the component's APIs.

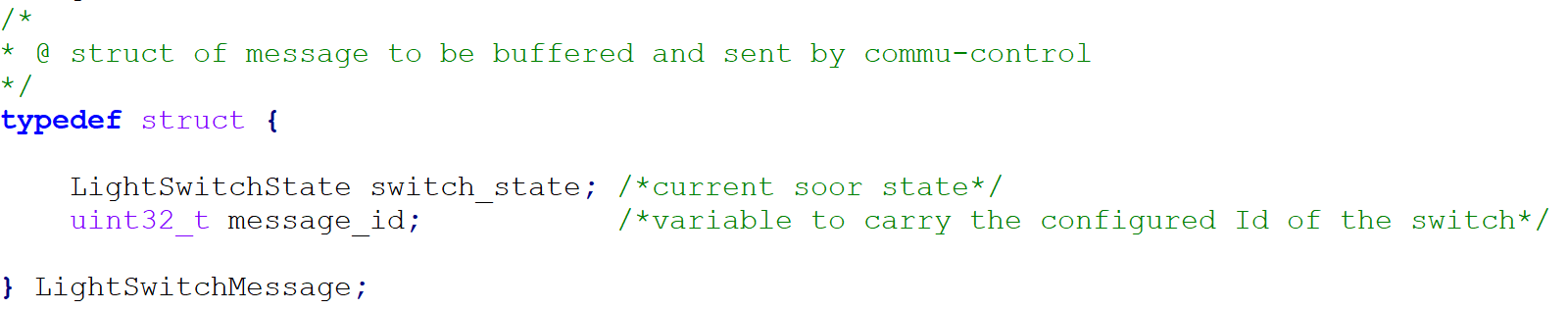
**API type to identify the ID for the message:**



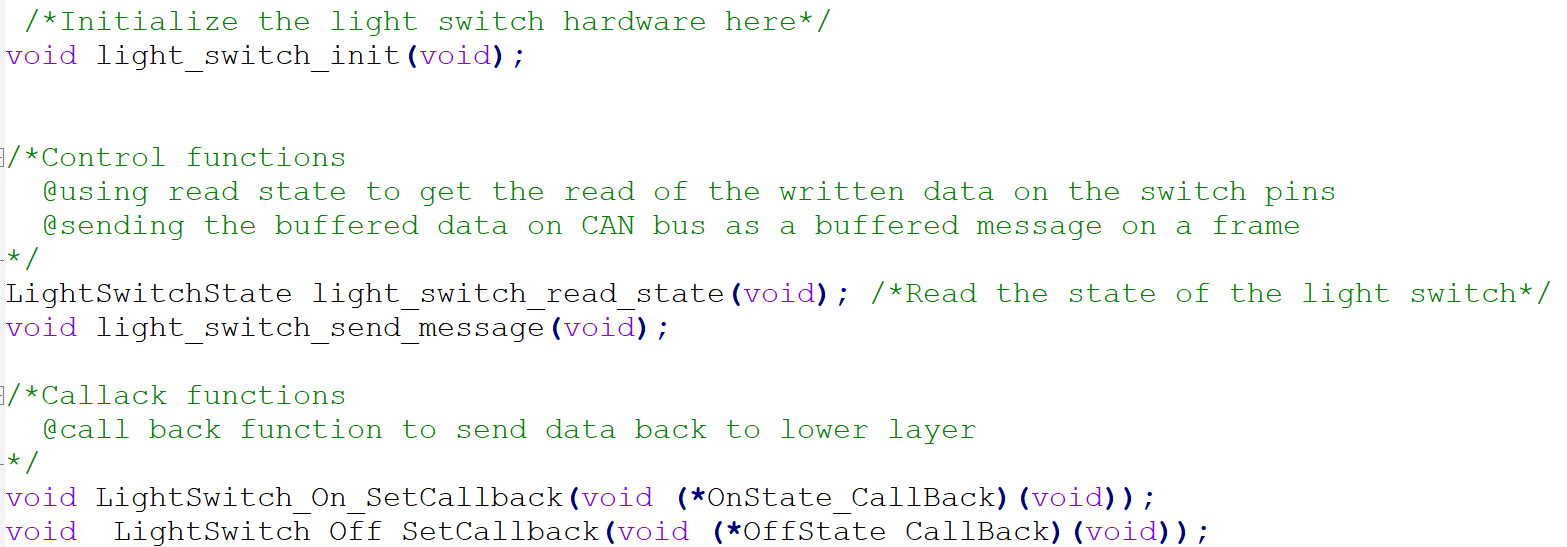
**API type to identify the current state of the switch:**

****

**API Type used to struct the message sent by light switch:**



**API function to initialize the component and get switches state as well as set the callback functions that called on changing the state:**



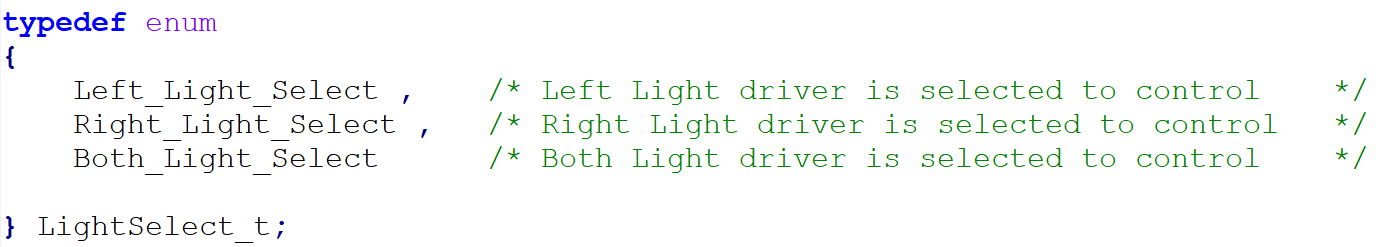
**ECU 2 Components:**

This ECU 2 will feature two light drivers, one each for the left and right lights, as well as a gate to control an alarm or buzzer. To simplify the control of these lights, they will be combined into one module component, with APIs provided to manage them both at once.

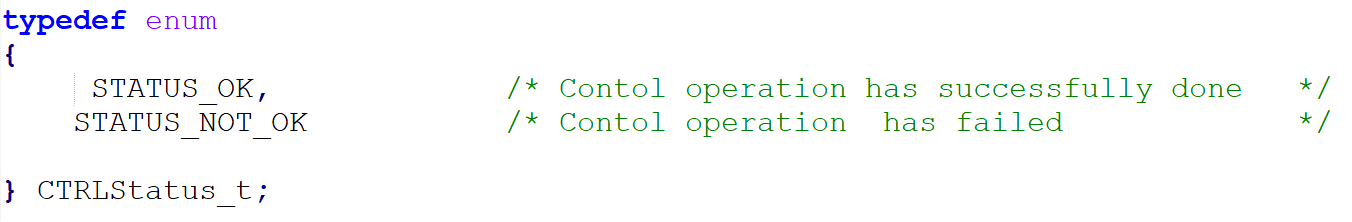
1. Light control

Here we define types and API functions to control on both of the right and left lights.

**API types that used in selecting between the two drivers to control:**

****

**API type used to provide status of the performed operation:**

****

**API functions that used to initialization and control operations:**

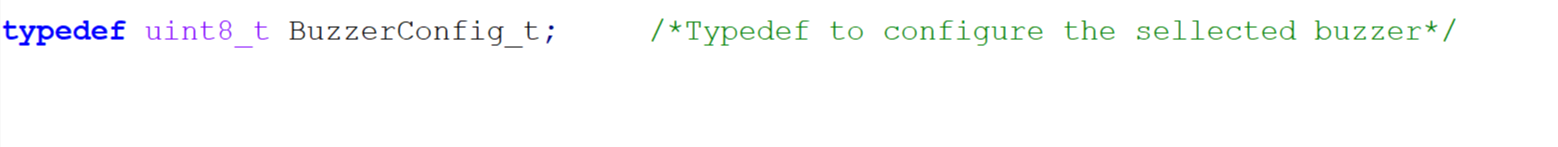
**Text, letter

Description automatically generated**

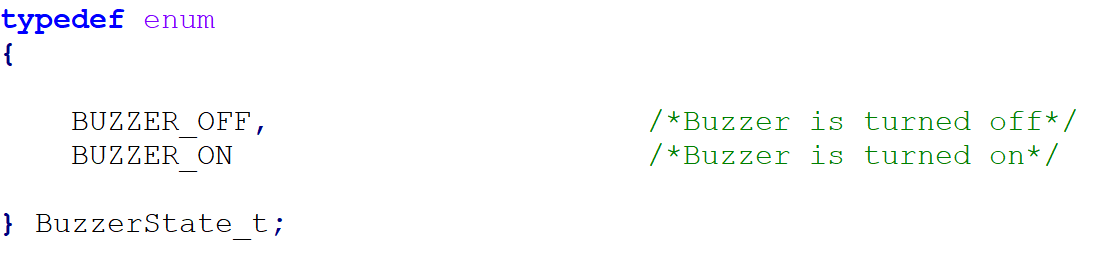
1. Alarm Control:

Here we define the APIs used to Control the Alarm operations.

**API type used to identify the selected Buzzer operation:**



**API type used to identify the performed operation condition:**

****

**API function to initialize the module and control operations:**

**Text

Description automatically generated**