**Project (3)**

|  |  |
| --- | --- |
| Name | Abdurrahman Mohame Elhefnawy |
| Email | Abdurrahman.elhefnwy@gmail.com |

**Automotive door control system design**

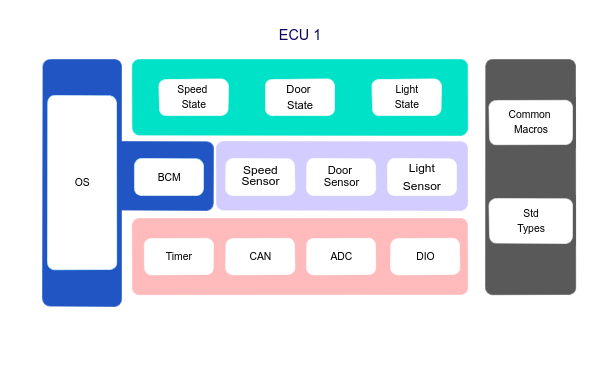
Block Diagram

Diagram

Description automatically generated

# ECU 1

# layered architecture



# DIO Module

# This module is for reading or writing from/to DIO pins.

# DIO Module’s APIs:

DIO\_init():

|  |  |
| --- | --- |
| Syntax | Void DIO\_init(void) |
| Description | This function used to initialize DIO module. |
| API Type | Init |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

DIO Module’s APIs:

DIO\_Write():

|  |  |
| --- | --- |
| Syntax | Void DIO\_Write(DIO\_PinType,Dio\_PinStateType) |
| Description | Function used to Write a state on a pin. |
| API Type | Setter |
| Parameters (in) | PinID , PinState |
| Parameters (out) | None |
| Return value | None |

DIO Module’s APIs:

DIO\_read():

|  |  |
| --- | --- |
| Syntax | Dio\_PinStateType DIO\_read(DIO\_PinType) |
| Description | Function used to the state of a pin. |
| API Type | Getter |
| Parameters (in) | PinID |
| Parameters (out) | None |
| Return value | PinState |

DIO Module’s typedefs:

|  |  |
| --- | --- |
| Type Name | Dio\_PinStateType |
| Description | Type used to describe pin state |
| Type | enum |
| Range | HIGH-LOW |

|  |  |
| --- | --- |
| Type Name | Dio\_PinType |
| Description | Type used to describe Pin Number |
| Type | enum |
| Range | PIN0 to PIN7 |

DIO Module’s typedefs:

|  |  |
| --- | --- |
| Type Name | Dio\_PinDirectionType |
| Description | Type used to describe Pin Direction |
| Type | enum |
| Range | INPUT- OUTPUT |

Timer Module

This module is for tick calculations and scheduling.

Timer Module’s APIs:

Timer\_init():

|  |  |
| --- | --- |
| Syntax | void Gpt\_Init ( ) |
| Description | Used to initialize Timers Module. |
| API Type | Init |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

Timer Module’s APIs:

Timer\_ StartTimer():

|  |  |
| --- | --- |
| Syntax | void Gpt\_StartTimer ( Gpt\_ChannelType Channel, Gpt\_Value ) |
| Description | Starts selected timer cahnnel with defined target time. |
| API Type | - |
| Parameters (in) | Channel: timer channel , Value: target counter value |
| Parameters (out) | None |
| Return value | None |

Timer Module’s APIs:

Timer\_ Gpt\_GetTimeElapsed ():

|  |  |
| --- | --- |
| Syntax | Gpt\_ValueType Gpt\_GetTimeElapsed ( Gpt\_ChannelType Channel ) |
| Description | Gets the Elapsed time of a specific channel |
| API Type | Getter |
| Parameters (in) | Channel: timer channel |
| Parameters (out) | None |
| Return value | Time Elapsed |

Timer Module’s typedefs:

|  |  |
| --- | --- |
| Type Name | GPT\_ValueType |
| Description | Type for holding timer ticks |
| Type | Uint8 |

|  |  |
| --- | --- |
| Type Name | Gpt\_ChannelType |
| Description | Type used to describe Channel Number |
| Type | enum |

ADC Module

This module is for Converting Analog readings into digital.

ADC Module’s APIs:

ADC\_Init()

:

|  |  |
| --- | --- |
| Syntax | void ADC\_Init(void) |
| Description | Used to initialize ADC Module. |
| API Type | Init |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

ADC Module’s APIs:

ADC\_read():

|  |  |
| --- | --- |
| Syntax | void ADC\_ read ( ADC\_ChannelType) |
| Description | Reads an analog voltage from a specific channel |
| API Type | - |
| Parameters (in) | Channel: timer channel , Value: target counter value |
| Parameters (out) | None |
| Return value | None |

ADC Module’s typedefs:

|  |  |
| --- | --- |
| Type Name | ADC\_ChannelType |
| Description | Type used to describe Channel Number |
| Type | enum |

CAN Module

This module is for Communication between the 2 ECUs.

CAN Module’s APIs:

CAN\_init():

|  |  |
| --- | --- |
| Syntax | void CAN\_Init ( ) |
| Description | Used to initialize CAN Module. |
| API Type | Init |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

CAN\_Module’s APIs:

CAN\_ SendData():

|  |  |
| --- | --- |
| Syntax | void CAN\_ SendByte ( Uint8 Data ) |
| Description | Sends one byte. |
| API Type | - |
| Parameters (in) | Data: One Byte |
| Parameters (out) | None |
| Return value | None |

CAN\_Module’s APIs:

CAN\_ SendString():

|  |  |
| --- | --- |
| Syntax | void CAN\_ SendString ( Uint8\* Str) |
| Description | Sends stream of byte. |
| API Type | - |
| Parameters (in) | Str: Stream of Bytes |
| Parameters (out) | None |
| Return value | None |

CAN\_Module’s APIs:

CAN\_ RecicveByte():

|  |  |
| --- | --- |
| Syntax | Uint8 CAN\_ RecieveByte ( void) |
| Description | Receives one byte. |
| API Type | - |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | Received Byte |

Light\_Sensor Module

This module is for Interfacing with the Light sensor.

Light\_Sensor Module’s APIs:

Light\_Sensor \_init():

|  |  |
| --- | --- |
| Syntax | Light\_Sensor \_init(): |
| Description | Used to initialize Light\_Sensor Module. |
| API Type | Init |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

Light\_Sensor Module’s APIs:

Light\_Sensor \_ReadValue():

|  |  |
| --- | --- |
| Syntax | Uint\_8 Light\_Sensor \_ReadValue(void): |
| Description | Used to read light sensor data. |
| API Type | Getter |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | Sensor Value |

Door\_Sensor Module

This module is for Interfacing with the Door sensor.

Door\_Sensor Module’s APIs:

Light\_Sensor \_init():

|  |  |
| --- | --- |
| Syntax | Door\_Sensor \_init(): |
| Description | Used to initialize Door\_Sensor Module. |
| API Type | Init |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

Door\_Sensor Module’s APIs:

Door\_Sensor \_ReadValue():

|  |  |
| --- | --- |
| Syntax | Uint\_8 Door\_Sensor \_ReadValue(void): |
| Description | Used to read Door sensor data. |
| API Type | Getter |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | Sensor Value |

Speed\_Sensor Module

This module is for Interfacing with the Speed sensor.

Speed \_Sensor Module’s APIs:

Light\_Sensor \_init():

|  |  |
| --- | --- |
| Syntax | Speed\_Sensor \_init(): |
| Description | Used to initialize Speed\_Sensor Module. |
| API Type | Init |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

Speed\_Sensor Module’s APIs:

Speed\_Sensor \_ReadValue():

|  |  |
| --- | --- |
| Syntax | Uint\_8 Speed\_Sensor \_ReadValue(void): |
| Description | Used to read Speed sensor data. |
| API Type | Getter |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | Sensor Value |

BCM Module

This module is for managing Communication process.

BCM Module’s APIs:

BCM \_init():

|  |  |
| --- | --- |
| Syntax | BCM\_init(): |
| Description | Used to initialize BCM Module. |
| API Type | Init |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

BCM Module’s APIs:

BCM\_SendString():

|  |  |
| --- | --- |
| Syntax | Void BCM\_SendString(BusType bus,uint8\* Str): |
| Description | Used to send string. |
| API Type | - |
| Parameters (in) | Bus Type and string to be sent |
| Parameters (out) | None |
| Return value | None |

BCM Module’s APIs:

BCM\_Recieve():

|  |  |
| --- | --- |
| Syntax | Uint8 BCM\_ReceiveBusType bus): |
| Description | Used to receive byte. |
| API Type | Getter |
| Parameters (in) | Bus Type |
| Parameters (out) | None |
| Return value | Received byte. |

BCM Module’s typedefs:

|  |  |
| --- | --- |
| Type Name | BCM\_BusType |
| Description | Type used to describe Bus Id |
| Type | enum |

Light\_State Module

This module is for reading from light module and send its state.

Light\_State Module’s APIs:

Light\_State\_monitor ():

|  |  |
| --- | --- |
| Syntax | Void Light\_State\_monitor(): |
| Description | Used for for reading from light module and send its state. |
| API Type | - |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

Door\_State Module

This module is for reading from Door modue and send its state.

Door\_State Module’s APIs:

Door\_State\_monitor ():

|  |  |
| --- | --- |
| Syntax | Void Door\_State\_monitor(): |
| Description | Used for for reading from Door module and send its state. |
| API Type | - |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

Speed\_State Module

This module is for reading from Speed module and send its state.

Speed\_State Module’s APIs:

Speed\_State\_monitor ():

|  |  |
| --- | --- |
| Syntax | Void Speed\_State\_monitor(): |
| Description | Used for for reading from Speed modue and send its state. |
| API Type | - |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

# ECU 2

# layered architecture

Graphical user interface

Description automatically generated

# DIO Module

# This module is for reading or writing from/to DIO pins.

# DIO Module’s APIs:

DIO\_init():

|  |  |
| --- | --- |
| Syntax | Void DIO\_init(void) |
| Description | This function used to initialize DIO module. |
| API Type | Init |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

DIO Module’s APIs:

DIO\_Write():

|  |  |
| --- | --- |
| Syntax | Void DIO\_Write(DIO\_PinType,Dio\_PinStateType) |
| Description | Function used to Write a state on a pin. |
| API Type | Setter |
| Parameters (in) | PinID , PinState |
| Parameters (out) | None |
| Return value | None |

DIO Module’s APIs:

DIO\_read():

|  |  |
| --- | --- |
| Syntax | Dio\_PinStateType DIO\_read(DIO\_PinType) |
| Description | Function used to the state of a pin. |
| API Type | Getter |
| Parameters (in) | PinID |
| Parameters (out) | None |
| Return value | PinState |

DIO Module’s typedefs:

|  |  |
| --- | --- |
| Type Name | Dio\_PinStateType |
| Description | Type used to describe pin state |
| Type | enum |
| Range | HIGH-LOW |

|  |  |
| --- | --- |
| Type Name | Dio\_PinType |
| Description | Type used to describe Pin Number |
| Type | enum |
| Range | PIN0 to PIN7 |

DIO Module’s typedefs:

|  |  |
| --- | --- |
| Type Name | Dio\_PinDirectionType |
| Description | Type used to describe Pin Direction |
| Type | enum |
| Range | INPUT- OUTPUT |

Timer Module

This module is for tick calculations and scheduling.

Timer Module’s APIs:

Timer\_init():

|  |  |
| --- | --- |
| Syntax | void Gpt\_Init ( ) |
| Description | Used to initialize Timers Module. |
| API Type | Init |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

Timer Module’s APIs:

Timer\_ StartTimer():

|  |  |
| --- | --- |
| Syntax | void Gpt\_StartTimer ( Gpt\_ChannelType Channel, Gpt\_Value ) |
| Description | Starts selected timer cahnnel with defined target time. |
| API Type | - |
| Parameters (in) | Channel: timer channel , Value: target counter value |
| Parameters (out) | None |
| Return value | None |

Timer Module’s APIs:

Timer\_ Gpt\_GetTimeElapsed ():

|  |  |
| --- | --- |
| Syntax | Gpt\_ValueType Gpt\_GetTimeElapsed ( Gpt\_ChannelType Channel ) |
| Description | Gets the Elapsed time of a specific channel |
| API Type | Getter |
| Parameters (in) | Channel: timer channel |
| Parameters (out) | None |
| Return value | Time Elapsed |

Timer Module’s typedefs:

|  |  |
| --- | --- |
| Type Name | GPT\_ValueType |
| Description | Type for holding timer ticks |
| Type | Uint8 |

|  |  |
| --- | --- |
| Type Name | Gpt\_ChannelType |
| Description | Type used to describe Channel Number |
| Type | enum |

CAN Module

This module is for Communication between the 2 ECUs.

CAN Module’s APIs:

CAN\_init():

|  |  |
| --- | --- |
| Syntax | void CAN\_Init ( ) |
| Description | Used to initialize CAN Module. |
| API Type | Init |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

CAN\_Module’s APIs:

CAN\_ SendData():

|  |  |
| --- | --- |
| Syntax | void CAN\_ SendByte ( Uint8 Data ) |
| Description | Sends one byte. |
| API Type | - |
| Parameters (in) | Data: One Byte |
| Parameters (out) | None |
| Return value | None |

CAN\_Module’s APIs:

CAN\_ SendString():

|  |  |
| --- | --- |
| Syntax | void CAN\_ SendString ( Uint8\* Str) |
| Description | Sends stream of byte. |
| API Type | - |
| Parameters (in) | Str: Stream of Bytes |
| Parameters (out) | None |
| Return value | None |

CAN\_Module’s APIs:

CAN\_ RecicveByte():

|  |  |
| --- | --- |
| Syntax | Uint8 CAN\_ RecieveByte ( void) |
| Description | Receives one byte. |
| API Type | - |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | Received Byte |

Buzzer Module

This module is for Interfacing with the Buzzer.

Buzzer Module’s APIs:

Buzzer\_init():

|  |  |
| --- | --- |
| Syntax | Void Buzzer\_init(void): |
| Description | Used to initialize Buzzer Module. |
| API Type | Init |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

Buzzer Module’s APIs:

Buzzer\_On():

|  |  |
| --- | --- |
| Syntax | Void Buzzer\_On(void): |
| Description | Used to turn the buzzer on . |
| API Type | - |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

Buzzer Module’s APIs:

Buzzer\_Off():

|  |  |
| --- | --- |
| Syntax | Void Buzzer\_Off(void): |
| Description | Used to turn the buzzer Off. |
| API Type | - |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

Right\_Light Module

This module is for Interfacing with the Right light.

Right\_Light Module’s APIs:

Right\_Light\_init():

|  |  |
| --- | --- |
| Syntax | Void Right\_Light \_init(void): |
| Description | Used to initialize Right\_Light Module. |
| API Type | Init |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

Right\_Light Module’s APIs:

Right\_Light \_On():

|  |  |
| --- | --- |
| Syntax | Void Right\_Light \_On(void): |
| Description | Used to turn the Right\_Light on . |
| API Type | - |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

Right\_Light Module’s APIs:

Right\_Light \_Off():

|  |  |
| --- | --- |
| Syntax | Void Right\_Light \_Off(void): |
| Description | Used to turn the Right\_Light Off. |
| API Type | - |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

Left\_Light Module

This module is for Interfacing with the Left light.

Left \_Light Module’s APIs:

Left \_Light\_init():

|  |  |
| --- | --- |
| Syntax | Void Left \_Light \_init(void): |
| Description | Used to initialize Left \_Light Module. |
| API Type | Init |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

Left \_Light Module’s APIs:

Left \_Light \_On():

|  |  |
| --- | --- |
| Syntax | Void Left \_Light \_On(void): |
| Description | Used to turn the Left \_Light on . |
| API Type | - |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

Left \_Light Module’s APIs:

Left \_Light \_Off():

|  |  |
| --- | --- |
| Syntax | Void Left \_Light \_Off(void): |
| Description | Used to turn the Left \_Light Off. |
| API Type | - |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

BCM Module

This module is for managing Communication process.

BCM Module’s APIs:

BCM \_init():

|  |  |
| --- | --- |
| Syntax | BCM\_init(): |
| Description | Used to initialize BCM Module. |
| API Type | Init |
| Parameters (in) | None |
| Parameters (out) | None |
| Return value | None |

BCM Module’s APIs:

BCM\_SendString():

|  |  |
| --- | --- |
| Syntax | Void BCM\_SendString(BusType bus,uint8\* Str): |
| Description | Used to send string. |
| API Type | - |
| Parameters (in) | Bus Type and string to be sent |
| Parameters (out) | None |
| Return value | None |

BCM Module’s APIs:

BCM\_Recieve():

|  |  |
| --- | --- |
| Syntax | Uint8 BCM\_ReceiveBusType bus): |
| Description | Used to receive byte. |
| API Type | Getter |
| Parameters (in) | Bus Type |
| Parameters (out) | None |
| Return value | Received byte. |

BCM Module’s typedefs:

|  |  |
| --- | --- |
| Type Name | BCM\_BusType |
| Description | Type used to describe Bus Id |
| Type | enum |

Buzzer\_Controller Module

This module is for Controlling Buzzer module.

Buzzer\_Controller Module’s APIs:

Buzzer\_Control():

|  |  |
| --- | --- |
| Syntax | Void Buzzer\_Controll(Buzzer\_stateType state ): |
| Description | Used for for Controlling Buzzer module. |
| API Type | - |
| Parameters (in) | State ON or OFF |
| Parameters (out) | None |
| Return value | None |

Buzzer Controller Module’s typedefs:

|  |  |
| --- | --- |
| Type Name | Buzzer\_stateType |
| Description | Type used to describe Buzzer State |
| Type | enum (HIGH-LOW) |

LL\_Controller Module

This module is for Controlling LLmodule.

LL\_Controller Module’s APIs:

LL\_Control():

|  |  |
| --- | --- |
| Syntax | Void LL\_Controll(LL\_stateType state ): |
| Description | Used for for Controlling LL module. |
| API Type | - |
| Parameters (in) | State ON or OFF |
| Parameters (out) | None |
| Return value | None |

LL Controller Module’s typedefs:

|  |  |
| --- | --- |
| Type Name | LL\_stateType |
| Description | Type used to describe LL State |
| Type | enum (HIGH-LOW) |

RL\_Controller Module

This module is for Controlling RL module.

RL\_Controller Module’s APIs:

RL\_Control():

|  |  |
| --- | --- |
| Syntax | Void RL\_Controll(RL\_stateType state ): |
| Description | Used for for Controlling RL module. |
| API Type | - |
| Parameters (in) | State ON or OFF |
| Parameters (out) | None |
| Return value | None |

RL Controller Module’s typedefs:

|  |  |
| --- | --- |
| Type Name | RL\_stateType |
| Description | Type used to describe RL State |
| Type | enum (HIGH-LOW) |