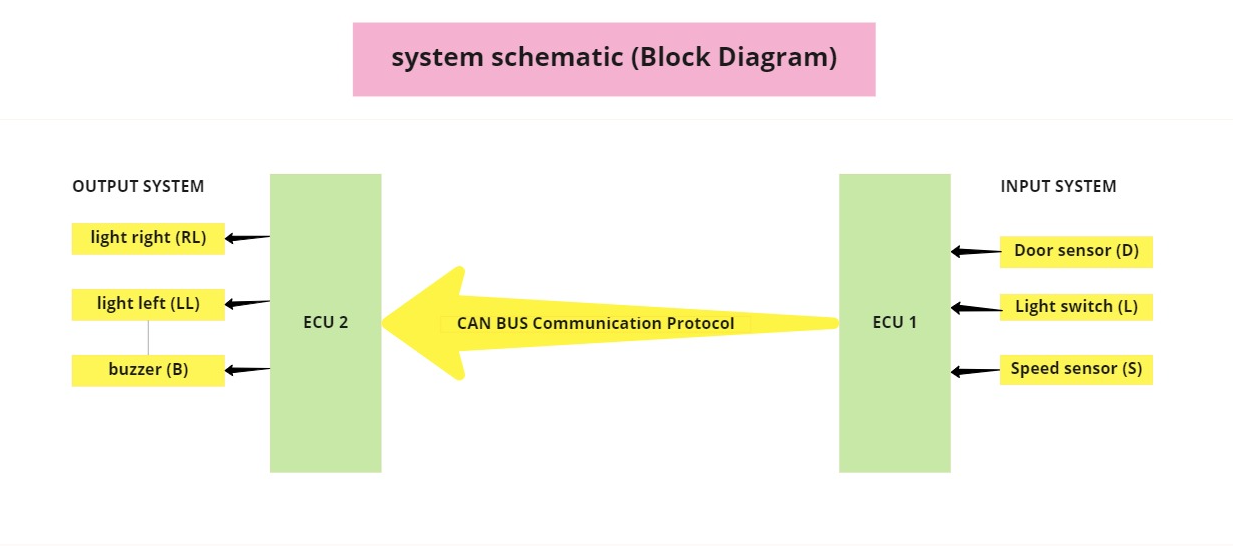
# Automotive door control system design Static Design Report

**Name: Hossam El-Deen Medhat Hussein Ebrahim**

**Email:** [hossam.medhat998@gmail.com](mailto:hossam.medhat998@gmail.com)

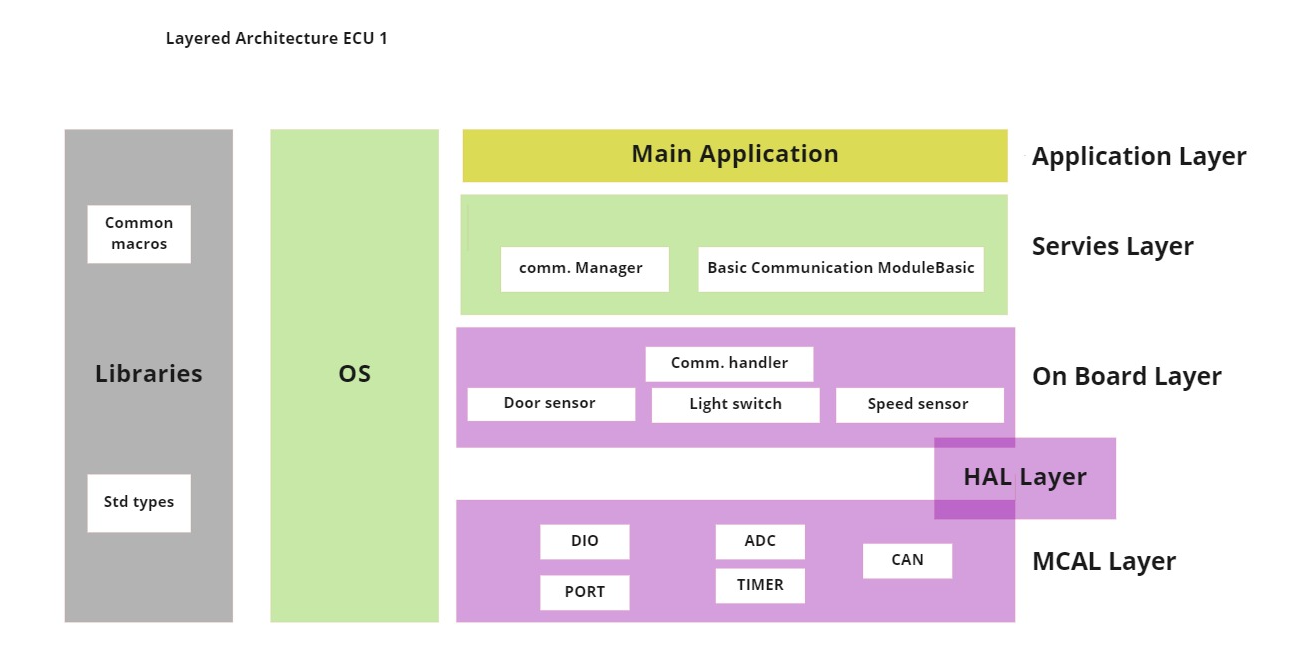
**System Schematic (Block Diagram)**



# Static Design:

## For ECU 1:

### The Layered Architecture:



* 1. **ECU components and modules Components Connected:**

1. CAN BUS Communication Protocol (for communication between the two ECUs)
2. Light switch
3. Speed Sensor
4. Door Sensor

### Modules:

#### External hardware:

1. CAN transiver module
2. Switch module
3. Speed Sensor module
4. Door Sensor module

#### Internal hardware:

1. Port Module (initialize all pins required with modes)
2. DIO Module (switch module, Door Sensor module)
3. TIMER module (timer for application)
4. ADC module (for speed sensor)
5. CAN Module (for can transiver data )

### APIs

* 1. **Tasks in Application Layer**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Layer | Module | APIs | API Details | |
| Application Layer | Main Application | DoorSensorTask |  | |
| **Syntax:** | void DoorSensorTask(void); |
| **Sync/Async:** | Synchronous |
| **Reentrancy:** | Non-Reentrant |
| **Parameters:** | None |
| **Return:** | None |
| **Description:** | Manage Door Sensor Task |
|  | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Layer | Module | APIs | API Details | |
| Application Layer | Main Application | LightSwitchTask  SpeedSensorTask |  | |
| **Syntax:** | void LightSwitchTask(void); |
| **Sync/Async:** | Synchronous |
| **Reentrancy:** | Non-Reentrant |
| **Parameters:** | None |
| **Return:** | None |
| **Description:** | Manage Light Switch Task |
|  | |
| **Syntax:** | void SpeedSensorTask(void); |
| **Sync/Async:** | Synchronous |
| **Reentrancy:** | Non-Reentrant |
| **Parameters:** | None |
| **Return:** | None |
| **Description:** | Manage Speed Sensor Task |
|  | |

* 1. **Modules in Service Layer:**

|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Module | APIs | API Details |
| Servies Layer | Basic Communication ModuleBasic  (BCM Manager) | BCM\_Manager | |  |  | | --- | --- | | Syntax: | void BCM\_Manager (uint8\_t Id\_Bus, uint64\_t Data ); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | **Id\_Bus**: that the ID commutation protocol want to connect it,  **Data** :that the data want to send by BCM manager | | Return: | None | | Description: | Manage request the data Transmitter by CAN Bus W.R.T Id Bus selection | |
| Servies Layer | comm. Manager | Sensor\_Manager  (do Monitoring Sensors) | |  |  | | --- | --- | | Syntax: | Level\_States Sensor\_Manager (Id\_sensor Id\_Sensor\_read); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | **Id\_Sensor\_read** : that id Sensor selection want to read states | | Return: | Date of states Read from sensor | | Description: | Manage request read states of data from sensor selection | |

Types define of argument of APIs:

|  |  |
| --- | --- |
| Types | Define |
| typedef unsigned char uint8\_t | Used in armament Id\_Bus to select bus connect  range{0,255 } that range depended commutation to managed by BCM ,size 8bit |
| typedef unsigned long long uint64\_t | used because max width of data in CAN frame is 64 bits and used in argument Data transmitter API  BCM\_manager and Handler |
| Level\_States | typedef enum {Low, High } Level\_States  range{0,1} size 1bit |
| Id\_sensor | typedef enum {Sensor\_1, sensor\_2, sensor\_3} Id\_sensor  range{0,2 max sensor in project } size 2 bit |

* 1. **Modules in On Board Layer**

|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Module | APIs | API Details |
| On Board Layer | Comm. Handler | BCM\_Handler | |  |  | | --- | --- | | Syntax: | void BCM\_ Handler (uint8\_t Id\_Bus, uint64\_t Data ); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | **Id\_Bus**: that the ID commutation protocol want to connect it,  **Data** :that the data want to send by BCM manager | | Return: | None | | Description: | Manage request the data Transmitter by CAN Bus W.R.T Id Bus selection but deals with Hardware directly | |
| On Board Layer | Comm. Handler | Sensor Handler | |  |  | | --- | --- | | Syntax: | Level\_States Sensor\_Handler (Id\_sensor Id\_Sensor\_read); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | **Id\_Sensor\_read** : that id Sensor selection want to read states | | Return: | Date of states Read from sensor | | Description: | Manage request read states of data from sensor selection but deals with Hardware directly | |
| On Board Layer | Door Sensor | DoorSensor\_Init  DoorSensor\_ReadStatus | |  |  | | --- | --- | | Syntax: | void DoorSensor\_Init (void); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | None | | Return: | None | | Description: | Initialize the used DIO pins for digital input |  |  |  | | --- | --- | | Syntax: | Status\_door DoorSensor\_ReadStatus (void); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | None | | Return: | Status of the sensor door closed or opened | | Description: | Get the status of the sensor door  (closed or not ) | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| On Board Layer | Light Switch | LightSwitch\_Init  LightSwitch\_ReadStatus | |  |  | | --- | --- | | Syntax: | Void LightSwitch\_Init (void); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | None | | Return: | None | | Description: | Initialize the used DIO pins for digital input |  |  |  | | --- | --- | | Syntax: | Status\_switch LightSwitch\_ReadStatus (void); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | None | | Return: | Status of the light switch  Pressed or unpressed ) | | Description: | Get the status of the Light Switch  (Pressed or unpressed ) | |
| On Board Layer | Speed Sensor | SpeedSensor\_Init  SpeedSensor\_ReadStatus | |  |  | | --- | --- | | Syntax: | void SpeedSensor\_Init (void); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | None | | Return: | None | | Description: | Initialize the used DIO pins for analog input For (ADC) |  |  |  | | --- | --- | | Syntax: | Status\_speed SpeedSensor\_ReadStatus (void); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | None | | Return: | Status of the sensor speed of car that can be moving or stopped | | Description: | Read the Status value of the speed sensor (moving or stop) | |

Types define of argument of APIs:

|  |  |
| --- | --- |
| Types | Define |
| typedef unsigned char uint8\_t | Used in armament Id\_Bus to select bus connect  range{0,255 } that range depended commutation to managed by BCM ,size 8bit |
| typedef unsigned long long uint64\_t | used because max width of data in CAN frame is 64 bits and used in argument Data transmitter API  BCM\_manager and Handler |
| Level\_States | typedef enum {Low, High} Level\_States  range{0,1} size 1bit |
| Id\_sensor | typedef enum {Sensor\_1, sensor\_2, sensor\_3} Id\_sensor  range{0,2 max sensor in project } size 2 bit |
| Status\_door | typedef enum {closed, opened} Status\_door  range{0,1} size 1bit |
| Status\_switch | typedef enum {undressed, pressed} Status\_switch  range{0,1} size 1bit |
| Status\_speed | typedef enum {stopped, moving} Status\_speed  range{0,1} size 1bit after convert value adc |

* 1. **Modules in MCAL Layer**

|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Module | APIs | API Details |
| MCAL Layer | DIO | DIO\_Init  DIO\_ReadChannel  DIO\_WriteChannel | |  |  | | --- | --- | | Syntax: | Void DIO\_Init (void); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | None | | Return: | None | | Description: | Initialize the used DIO pins with required configuration file . |  |  |  | | --- | --- | | Syntax: | LevelType DIO\_ReadChannel( Id\_channel channel); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | **Channel**: the value of channel want to read it the value of enum **Id\_channel** | | Return: | Status of pin High or low that value from Dio\_LevelType | | Description: | Read the channel required |  |  |  | | --- | --- | | Syntax: | void DIO\_WriteChannel (LevelType Level ); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | **Level** : Level want to write channel high level or low level | | Return: | None | | Description: | Write the level of the channel required | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MCAL Layer | PORT | Port\_init(\*Port\_cfg\_ptr) | |  |  | | --- | --- | | Syntax: | void Port\_init(\*Port\_cfg\_ptr) | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | This API takes pointer to the configuration container of the port driver to initialize the configured pins | | Return: | None | | Description: | Initialize the used Port with required configuration of the pointer | |
| MCAL Layer | PORT | void SetPinValue(port\_of\_Id port\_Id,Pin\_of\_num Pin\_num, Dio\_LevelType level ) | |  |  | | --- | --- | | Syntax: | void SetPinValue(port\_of\_Id port\_Id,Pin\_of\_num Pin\_num, Dio\_LevelType level ) | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | This API takes to the configuration **port\_Id** that type of port\_of\_Id to port\_1 or port\_2 …. , **Pin\_num** the number of pin want to configure, **level** that initiation of level of pin high or low | | Return: | None | | Description: | Initialize the used Port with required configuration of the Parameters. | |
| MCAL Layer  MCAL Layer | Timer  Timer | Timer\_Init  Timer\_Start  Timer\_Stop | |  |  | | --- | --- | | Syntax: | void Timer\_Init (void); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | None | | Return: | None | | Description: | Initialize timer required configuration |  |  |  | | --- | --- | | Syntax: | void Timer\_Start (timer\_ChannelType channel, timer\_ValueType value\_count ); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | **Channel**: that the channel wanted to start timer , **value\_count**  value of counter to count tick the mix value depend of over flow timer count | | Return: | None | | Description: | Initialize timer required configuration of Parameters to start count |  |  |  | | --- | --- | | Syntax: | Void Timer\_Stop (timer\_ChannelType channel); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | **Channel:** channel Id of timer wanted to stopped | | Return: | None | | Description: | Stop timer required configuration id channel | |
| MCAL Layer | CAN | CAN\_Init  CAN\_Transmiter | |  |  | | --- | --- | | Syntax: | void CAN\_Init (void); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | None | | Return: | None | | Description: | Initialize CAN bus required configuration and Hardware pin CAN |  |  |  | | --- | --- | | Syntax: | void CAN\_Transmiter (uint8\_t Pin\_Id,uint64\_t Data); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | Data transmitter by the can bus , Pin\_id the agreement to selection the id of bus wanted connected | | Return: | None | | Description: | Transmitter data by CAN Bus | |
| MCAL Layer | ADC | ADC\_Init  ADC\_ReadChannel | |  |  | | --- | --- | | Syntax: | void ADC\_Init (void); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | None | | Return: | None | | Description: | Initialize ADC required configuration and Hardware pin ADC connect speed sensor | |
|  |  |  | |  |  | | --- | --- | | Syntax: | uint16\_tADC\_ReadChannel(Pin\_of\_num Pin\_Id); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | Pin\_Id of ADC to read value | | Return: | The value of channel ADC | | Description: | Read the value of channel ADC | |
|  |  |  |  |

Types define of argument of APIs:

|  |  |
| --- | --- |
| Types | Define |
| LevelType | typedef enum {LOW, HIGH} Dio\_LevelType  range{0,1} size 1bit |
| Id\_channel | typedef enum {Channel\_1, Channel\_2, Channel\_3, Channel\_4, Channel\_5, Channel\_6, Channel\_7,  Channel\_8}Dio\_LevelType range{0,8} size 1bit |
| Port\_cfg\_ptr that of struct to configuration  Typedef struct{uint8\_t Port\_Pin\_Direction,  uint8\_t PORT\_PIN\_INTERNAL\_ATTACH, uint8\_t PORT\_PIN\_LEVEL\_VALUE ,  uint8\_t PORT\_def\_PORTx,  uint8\_t PORT\_def\_PINx,  uint8\_t PORT\_def\_Mode\_x}port\_config; | |  |  | | --- | --- | | Port\_Pin\_Direction | Used to set the direction input or output | | PORT\_PIN\_INTERNAL\_ATTACH | Used to select the internal resistance | | PORT\_PIN\_LEVEL\_VALUE | Used to specify the initial value | | PORT\_def\_PORTx | This typedef used to point to specific port , if x equal A then this is portA | | PORT\_def\_PINx | This typedef used to point to specific pin , if x equal 0 then this is pin0 | | PORT\_def\_Mode\_x | This typedef used to point to specific mode , if x equal adc then this is adc mode | |
| port\_of\_Id | typedef enum {Port\_1, Port \_2 Port \_3, Port l\_4, Port \_5, Port \_6 Port \_7} port\_of\_Id range{0,8} size 1bit |
| ,Pin\_of\_num | typedef enum {Pin\_1, Pin \_2 Pin \_3, Pin l\_4, Pin \_5, Pin\_6, Pin \_7,Pin\_8} Pin\_of\_num range{0,8} size 1bit |
| typedef uint32\_t T timer\_ValueType; | Value of tick range from 0 to 2^32 -1 size 32 bit |
| Typedef enum {T1 = T1PR,T2 = T2PR,Etc:}  timer\_ChannelType; | This enum types stores the identifier for the Channel like  its name. |

* 1. Folder Structure according to the previous points:

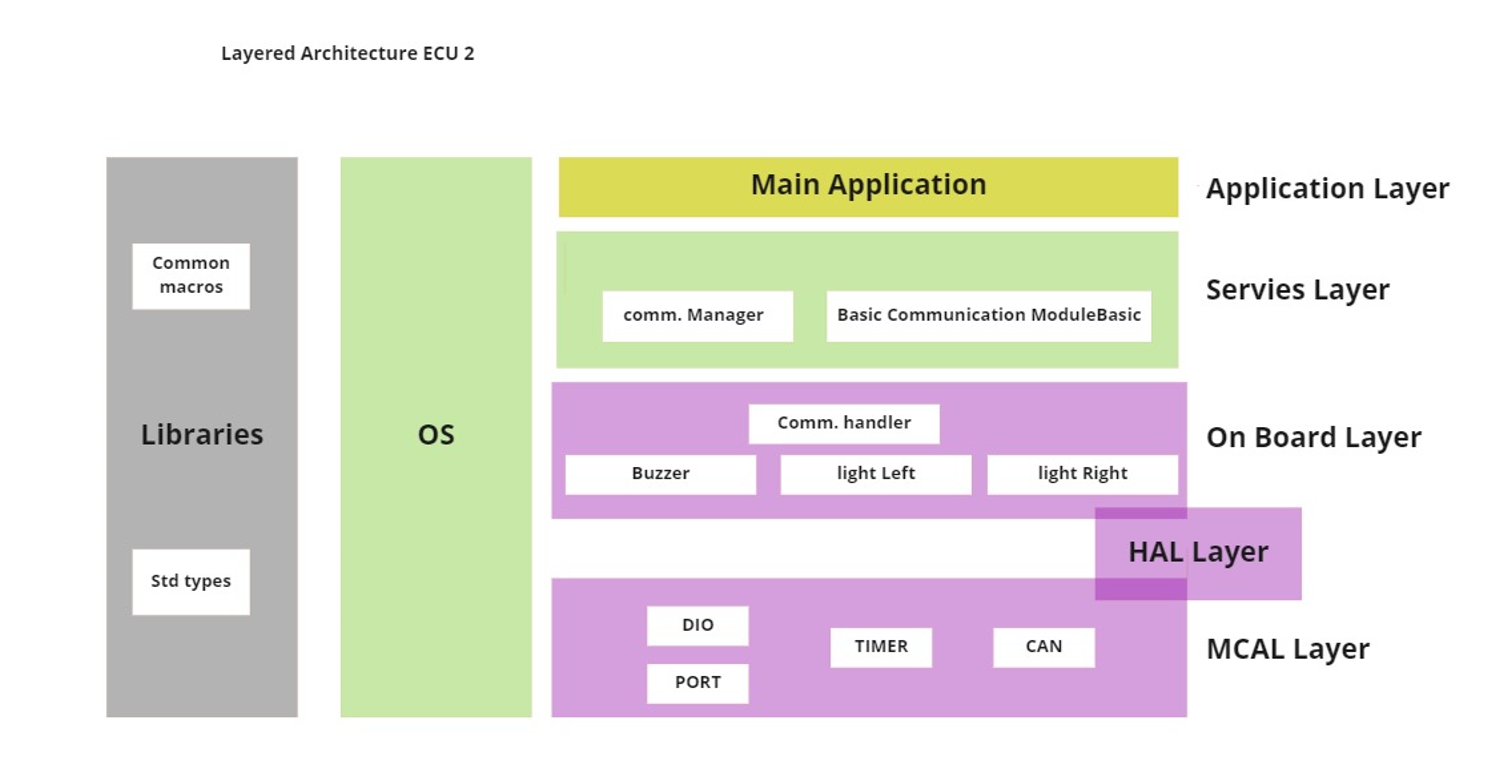
|  |  |  |
| --- | --- | --- |
| Application folder | Servies folder | On Board Layer |
| main.c | Operting\_system.c | BCM\_Handler.c |
|  | BCM\_Manager.c | Sensor\_Handler.c |
|  | Sensor\_Manager.c | Door\_sensor.c |
|  |  | Light\_switch.c |
|  |  | Speed\_sensor.c |

|  |  |
| --- | --- |
| MCAL folder | Configure folder |
| dio.c | Timer\_config.c |
| port.c | Adc\_config.c |
| adc.c | Can\_config.c |
| Timer.c | Port\_config.c |
| can.c | Dio\_config.c |
|  | Door\_sensorconfig.c |
|  | Light\_switchconfig.c |
|  | Speed\_sensorconfig.c |

|  |
| --- |
| Commen folder (all the header (name.h)) |
| Mainapp.h / os.h / servies.h |
| BCS\_manager.h/Sonser\_manager.h |
| Light\_switch.h / speed\_sonser.h / Door\_sensor.h |
| Dio.h / port.h / timer.h /can.h/adc.h |
| dio\_config.h/port\_config.h / timer\_config.h /can\_config.h /adc\_config.h |
| Stdtypes.h /comman\_macro.h /Hw.h |

## For ECU 2:

### The Layered Architecture:



1. **ECU Components & Modules Components Connected:**
2. CAN BUS Communication Protocol (for communication between the two ECUs)
3. Light right
4. Light left
5. Buzzer

### Modules:

#### External hardware:

1. CAN transiver module
2. Light left module
3. Light right module
4. Buzzer module

#### Internal hardware:

1. Port Module (initialize all pins required with modes)
2. DIO Module (switch module, Door Sensor module)
3. TIMER module (timer for application)
4. CAN Module (for can transiver data)

### APIs

* 1. **Modules in Application Layer**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Layer | Module | APIs | API Details | |
| Application Layer | Main Application | PeriodicReceive\_Status |  | |
| **Syntax:** | Void PeriodicReceive\_Status(uint64\_t \*  data ,uint8\_t\* id\_CAN); |
| **Sync/Async:** | Synchronous |
| **Reentrancy:** | Non-Reentrant |
| **Parameters:** | Pointer to data act as buffer for data  ,pointer of CAN bus id to id cheek it |
| **Return:** | None |
| **Description:** | Manage received data periodicity status of  ECU1 |

* 1. **Modules in Servies Layer**

|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Module | APIs | API Details |
| Servies Layer | Basic Communication ModuleBasic  (BCM Manager) | BCM\_Manager | |  |  | | --- | --- | | Syntax: | uint64\_t BCM\_Manager (uint8\_t Id\_Bus); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | **Id\_Bus**: that the ID commutation protocol want to connect it to received data | | Return: | Return Data frame of CAN bus that the data want to receive by CAN bus from ECU1 | | Description: | Manage request the data received by CAN Bus W.R.T Id Bus selection | |
| Servies Layer | comm. Manager | Actuator\_Manager (do Monitoring Action ) | |  |  | | --- | --- | | Syntax: | Void Actuator\_Manager ( actuator\_Id  actuator , action\_status\_action ); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | **actuator\_id** selection want to do action  states , **action** want to do(on ,off ) Actuator | | Return: | None | | Description: | Monitoring action request to do actuator selection | |

Types define of argument of APIs:

|  |  |
| --- | --- |
| Types | Define |
| typedef unsigned char uint8\_t | Used in armament Id\_Bus to select bus connect  range{0,255 } that range depended commutation to managed by BCM ,size 8bit |
| typedef unsigned long long uint64\_t | used because max width of data in CAN frame is 64 bits and used in argument Data received API  BCM\_manager and Handler |
| Status\_action | typedef enum {OFF,ON } status\_action  range{0,1} size 1bit |
| actuator\_Id | typedef enum { actuator\_1, actuator\_2} actuator\_Id  range{0,1} max actuator in project Buzzer and light } size 1 bit |

* 1. **Modules in On Board Layer**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Layer | Module | APIs | | API Details | |
| On Board Layer | Comm. Handler | BCM\_Handler | | |  |  | | --- | --- | | Syntax: | uint64\_t BCM\_Handler (uint8\_t Id\_Bus); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | **Id\_Bus**: that the ID commutation protocol want to connect it to received data | | Return: | Return Data frame of CAN bus that the data want to receive by CAN bus from ECU1 | | Description: | Handler request the data received by CAN Bus W.R.T Id Bus selection but deals with Hardware directly | | |
| On Board Layer | Comm. Handler | Actuator\_Handler | | |  |  | | --- | --- | | Syntax: | Void Actuator\_Handler ( actuator\_Id  actuator , action\_status\_action ); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | **actuator\_id** selection want to do action  states , **action** want to do(on ,off ) Actuator | | Return: | None | | Description: | Handler request to do action actuartor selection but deals with Hardware directly | | |
| On Board Layer | Buzzer | Buzzer\_Init  Buzzer\_on  Buzzer\_off |  | | | |
| **Syntax:** | | **Void Buzzer\_Init (void);** | |
| **Sync/Async:** | | Synchronous | |
| **Reentrancy:** | | Non-Reentrant | |
| **Parameters:** | | None | |
| **Return:** | | None | |
| **Description:** | | Initialize the used DIO pins for digital output respect to configuration | |
|  | | | |
| **Syntax:** | | **void Buzzer\_on(void);** | |
| **Sync/Async:** | | Synchronous | |
| **Reentrancy:** | | Non-Reentrant | |
| **Parameters:** | | None | |
| **Return:** | | None | |
| **Description:** | | Set Buzzer to turn on states | |
|  | | | |
| **Syntax:** | | **void Buzzer\_off(void);** | |
| **Sync/Async:** | | Synchronous | |
| **Reentrancy:** | | Non-Reentrant | |
| **Parameters:** | | None | |
| **Return:** | | None | |
| **Description:** | | Set Buzzer to turn off states | |
| On Board Layer | Light Switch | Light\_Init  Light\_off  Light\_on |  | | | |
| **Syntax:** | | **Void Light\_Init (void);** | |
| **Sync/Async:** | | Synchronous | |
| **Reentrancy:** | | Non-Reentrant | |
| **Parameters:** | | None | |
| **Return:** | | None | |
| **Description:** | | Initialize the used DIO pins for digital  output base the configuration | |
|  | | | |
| **Syntax:** | | **void Light\_off(void);** | |
| **Sync/Async:** | | Synchronous | |
| **Reentrancy:** | | Non-Reentrant | |
| **Parameters:** | | None | |
| **Return:** | | None | |
| **Description:** | | Set Light to turn off states | |
|  | | | |
| **Syntax:** | | **Void Light\_on(void);** | |
| **Sync/Async:** | | Synchronous | |
| **Reentrancy:** | | Non-Reentrant | |
| **Parameters:** | | None | |
| **Return:** | | None | |
| **Description:** | | Set light to turn on states | |

Types define of argument of APIs:

|  |  |
| --- | --- |
| Types | Define |
| typedef unsigned char uint8\_t | Used in armament Id\_Bus to select bus connect  range{0,255 } that range depended commutation to managed by BCM ,size 8bit |
| typedef unsigned long long uint64\_t | used because max width of data in CAN frame is 64 bits and used in argument Data received API  BCM\_manager and Handler |
| Status\_action | typedef enum {OFF,ON } status\_action  range{0,1} size 1bit |
| actuator\_Id | typedef enum { actuator\_1, actuator\_2} actuator\_Id  range{0,1} max actuator in project Buzzer and light } size 1 bit |

* 1. **Modules in MCAL Layer**

|  |  |  |  |
| --- | --- | --- | --- |
| Layer | Module | APIs | API Details |
| MCAL Layer | DIO | DIO\_Init  DIO\_ReadChannel  DIO\_WriteChannel | |  |  | | --- | --- | | Syntax: | Void DIO\_Init (void); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | None | | Return: | None | | Description: | Initialize the used DIO pins with required configuration file . |  |  |  | | --- | --- | | Syntax: | LevelType DIO\_ReadChannel( Id\_channel channel); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | **Channel**: the value of channel want to read it the value of enum **Id\_channel** | | Return: | Status of pin High or low that value from Dio\_LevelType | | Description: | Read the channel required |  |  |  | | --- | --- | | Syntax: | void DIO\_WriteChannel (LevelType Level ); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | **Level** : Level want to write channel high level or low level | | Return: | None | | Description: | Write the level of the channel required | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MCAL Layer | PORT | Port\_init(\*Port\_cfg\_ptr) | |  |  | | --- | --- | | Syntax: | void Port\_init(\*Port\_cfg\_ptr) | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | This API takes pointer to the configuration container of the port driver to initialize the configured pins | | Return: | None | | Description: | Initialize the used Port with required configuration of the pointer | |
| MCAL Layer | PORT | void SetPinValue(port\_of\_Id port\_Id,Pin\_of\_num Pin\_num, Dio\_LevelType level ) | |  |  | | --- | --- | | Syntax: | void SetPinValue(port\_of\_Id port\_Id,Pin\_of\_num Pin\_num, Dio\_LevelType level ) | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | This API takes to the configuration **port\_Id** that type of port\_of\_Id to port\_1 or port\_2 …. , **Pin\_num** the number of pin want to configure, **level** that initiation of level of pin high or low | | Return: | None | | Description: | Initialize the used Port with required configuration of the Parameters. | |
| MCAL Layer  MCAL Layer | Timer  Timer | Timer\_Init  Timer\_Start  Timer\_Stop | |  |  | | --- | --- | | Syntax: | void Timer\_Init (void); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | None | | Return: | None | | Description: | Initialize timer required configuration |  |  |  | | --- | --- | | Syntax: | void Timer\_Start (timer\_ChannelType channel, timer\_ValueType value\_count ); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | **Channel**: that the channel wanted to start timer , **value\_count**  value of counter to count tick the mix value depend of over flow timer count | | Return: | None | | Description: | Initialize timer required configuration of Parameters to start count |  |  |  | | --- | --- | | Syntax: | Void Timer\_Stop (timer\_ChannelType channel); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | **Channel:** channel Id of timer wanted to stopped | | Return: | None | | Description: | Stop timer required configuration id channel | |
| MCAL Layer | CAN | CAN\_Init  CAN\_ReceivedData | |  |  | | --- | --- | | Syntax: | void CAN\_Init (void); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | None | | Return: | None | | Description: | Initialize CAN bus required configuration and Hardware pin CAN |  |  |  | | --- | --- | | Syntax: | uint64\_t CAN\_ReceivedData (uint8\_t Pin\_Id); | | Sync/Async: | Synchronous | | Reentrancy: | Non-Reentrant | | Parameters: | **Pin\_id** the agreement to selection the id of bus wanted connected to received Data | | Return: | Data Received by the can bus | | Description: | Received data from CAN Bus | |

Types define of argument of APIs:

|  |  |
| --- | --- |
| Types | Define |
| LevelType | typedef enum {LOW, HIGH} Dio\_LevelType  range{0,1} size 1bit |
| Id\_channel | typedef enum {Channel\_1, Channel\_2, Channel\_3, Channel\_4, Channel\_5, Channel\_6, Channel\_7,  Channel\_8}Dio\_LevelType range{0,8} size 1bit |
| Port\_cfg\_ptr that of struct to configuration  Typedef struct{uint8\_t Port\_Pin\_Direction,  uint8\_t PORT\_PIN\_INTERNAL\_ATTACH, uint8\_t PORT\_PIN\_LEVEL\_VALUE ,  uint8\_t PORT\_def\_PORTx,  uint8\_t PORT\_def\_PINx,  uint8\_t PORT\_def\_Mode\_x}port\_config; | |  |  | | --- | --- | | Port\_Pin\_Direction | Used to set the direction input or output | | PORT\_PIN\_INTERNAL\_ATTACH | Used to select the internal resistance | | PORT\_PIN\_LEVEL\_VALUE | Used to specify the initial value | | PORT\_def\_PORTx | This typedef used to point to specific port , if x equal A then this is portA | | PORT\_def\_PINx | This typedef used to point to specific pin , if x equal 0 then this is pin0 | | PORT\_def\_Mode\_x | This typedef used to point to specific mode , if x equal adc then this is adc mode | |
| port\_of\_Id | typedef enum {Port\_1, Port \_2 Port \_3, Port l\_4, Port \_5, Port \_6 Port \_7} port\_of\_Id range{0,8} size 1bit |
| ,Pin\_of\_num | typedef enum {Pin\_1, Pin \_2 Pin \_3, Pin l\_4, Pin \_5, Pin\_6, Pin \_7,Pin\_8} Pin\_of\_num range{0,8} size 1bit |
| typedef uint32\_t T timer\_ValueType; | Value of tick range from 0 to 2^32 -1 size 32 bit |
| Typedef enum {T1 = T1PR, T2 = T2PR,Etc:}  timer\_ChannelType; | This enum types stores the identifier for the Channel like  its name. |

### Folder Structure according to the previous points:

|  |  |  |
| --- | --- | --- |
| Application folder | Servies folder | On Board Layer |
| main.c | Operting\_system.c | BCM\_Handler.c |
|  | BCM\_Manager.c | Actuator\_Handler.c |
|  | Actuator\_Manager.c | Buzzer\_sensor.c |
|  |  | Light.c |

|  |  |
| --- | --- |
| MCAL folder | Configure folder |
| dio.c | Timer\_config.c |
| port.c | Can\_config.c |
| can.c | Dio\_config.c |
| Timer.c | Port\_config.c |
|  | Light\_config.c |
|  | Buzzer\_config.c |

|  |
| --- |
| Commen folder (all the header (name.h)) |
| Mainapp.h / os.h / servies.h |
| BCS\_manager.h/ Actuator\_manager.h |
| Light\_.h / light.h |
| Dio.h / port.h / timer.h /can.h |
| dio\_config.h/port\_config.h / timer\_config.h /can\_config.h |