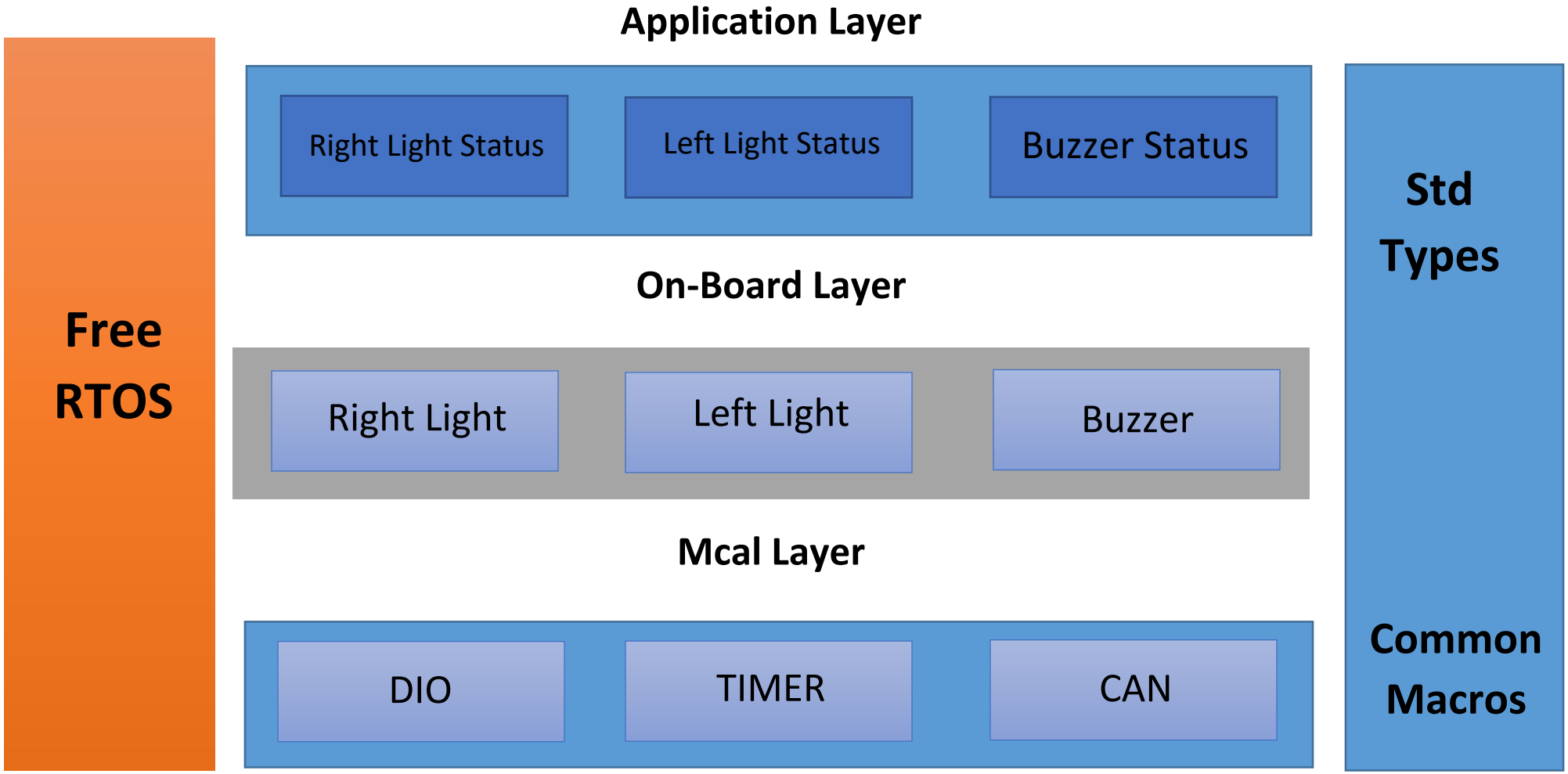
Automotive Door Control System Design (ECU2)



ECU1 has 3 abstraction layers: Application, Onboard and Mcal layers.

For application layer, it contains 3 main tasks:

1. Right Light status, which responsible for turning on or off the light according to the condition received from ECU1.
2. Left Light status, which responsible for turning on or off the light according to the condition received from ECU1.
3. Buzzer Status, which responsible for turning on or off the Buzzer according to the condition received from ECU1.

For On-Board layer, it contains 3 main tasks:

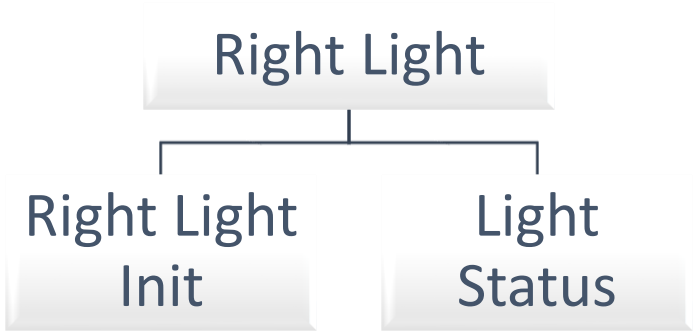
1. Right Light, hardware connected light to turn on or off.
2. Left Light, hardware connected light to turn on or off.
3. Buzzer, hardware connected Buzzer to turn on or off.

For Mcal layer, it contains 3 main tasks:

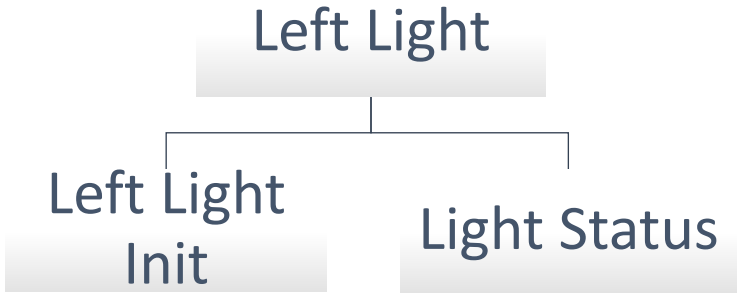
1. DIO, which responsible for logic output and input (Lights and Buzzer).
2. TIMER, which responsible for counting time in the system.
3. CAN, which responsible for communicating with another ECU and give the status of the components.

For APIs, that will be used in the Projects:

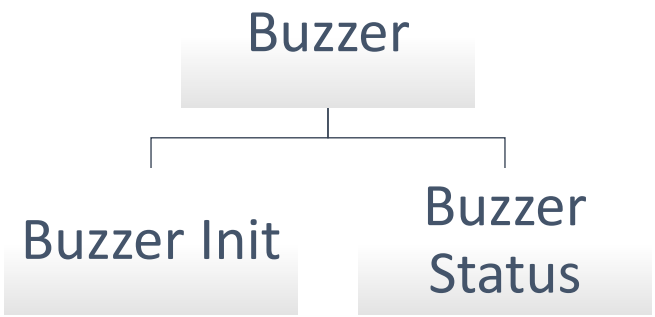
1. Right Light:



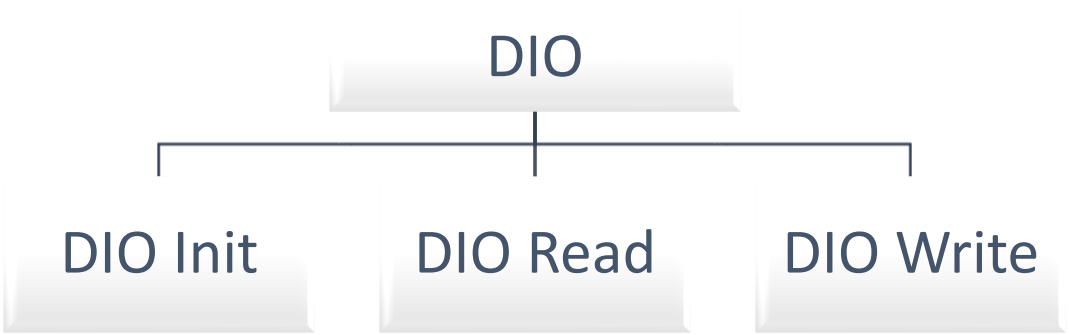
2. Left Light:



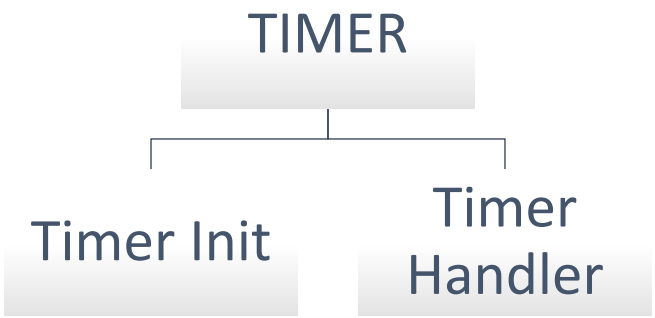
3. Buzzer:



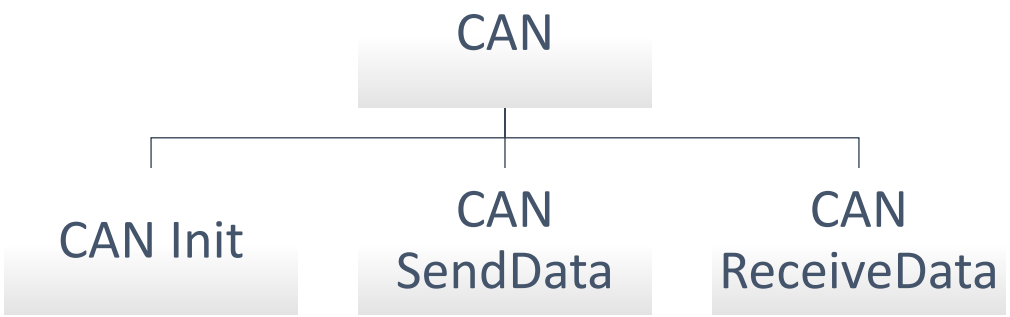
4. DIO:



5. TIMER:



6. CAN:



APIs Fully Detailed Description:

1. Right Light:

Must include DIO Driver.

API Arguments:

NAME	RIGHTLIGHTLAMP
TYPE	Enum
RANGE	0 for RightLightLamp_Off
	1 for RightLightLamp_On
DESCRIPTION	Describe if the pin is high or low.

API Functions:

void RightLight_Init (void);

Name	RightLight Init
API Type	Init
Arguments	void
Return Type	void
Description	that responsible for initialization of the Light module.

RightLightLamp Light_Status (Port PortNo , PinID PinNo, Pin PinStatus);

Name	**Light Status**	
API Type	Getter	
Arguments	Port	PortNo
	State which port is used	
	PinID	PinNo
	State which pin is used	
	Pin	PinStatus
	State the value of Pin (High or Low)	
Return Type	RightLightLamp	
Description	detect the status of the Light (On or Off).	

2. Left Light:

Must include DIO Driver.

API Arguments:

NAME	LEFTLIGHTLAMP
TYPE	Enum
RANGE	0 for LeftLightLamp_Off
	1 for LeftLightLamp_On
DESCRIPTION	Describe if the pin is high or low.

API Functions:

void LeftLight_Init (void);

Name	LeftLight Init
API Type	Init
Arguments	void
Return Type	void
Description	that responsible for initialization of the Light module.

LeftLightLamp Light_Status (Port PortNo , PinID PinNo, Pin PinStatus);

Name	**Light Status**	
API Type	Getter	
Arguments	Port	PortNo
	State which port is used	
	PinID	PinNo
	State which pin is used	
	Pin	PinStatus
	State the value of Pin (High or Low)	
Return Type	LeftLightLamp	
Description	detect the status of the Light (On or Off).	

3. Buzzer:

Must include DIO Driver.

API Arguments:

NAME	BUZZER
TYPE	Enum
RANGE	0 for Buzzer_Off
	1 for Buzzer_On
DESCRIPTION	Describe if the pin is high or low.

API Functions:

void Buzzer_Init(void);

Name	Buzzer Init
API Type	Init
Arguments	void
Return Type	void
Description	that responsible for initialization of the Buzzer module

Buzzer Buzzer_Status (Port PortNo , PinID PinNo, Pin PinStatus);

Name	**Buzzer Status**	
API Type	Getter	
Arguments	Port	PortNo
	State which port is used	
	PinID	PinNo
	State which pin is used	
	Pin	PinStatus
	State the value of Pin (High or Low)	
Return Type	Buzzer	
Description	detect the status of the Buzzer (On or Off).	

4. DIO:

API Arguments:

NAME	PIN
TYPE	Enum
RANGE	0 for PIN_IS_LOW
	1 for PIN_IS_HIGH
DESCRIPTION	Describe if the pin is high or low.

NAME	PORT
TYPE	Enum
RANGE	
DESCRIPTION	Describe which port is used.

NAME	PINNO
TYPE	Enum
RANGE	0 to 7 according to the No. of Pins Connected to Port (PIN0, PIN1,⋯)
DESCRIPTION	Describe which port is used.

NAME	DIO_CONFIGTYPE
TYPE	Structure
RANGE	Uint8
DESCRIPTION	Contain all configuration used to initialize the DIO port correctly. A pointer to structure is passed to the function with all information it needs.

API Functions:

void DIO_Init (DIO_ConfigType * ConfigStruct);

Name	DIO Init
API Type	Init
Arguments	DIO_ConfigType * ConfigStruct
	Structure for all configuration
Return Type	void
Description	initialize the DIO port with clock and determine which is input and output.

Pin DIO_Read (Port PortNo , PinID PinNo);

Name	**DIO Read**	
API Type	Getter	
Arguments	Port	PortNo
	State which port is used.	
	PinID	PinNo
	State which Pin used to get Data	
Return Type	Pin	
Description	responsible for reading the status of the pin.	

void DIO_Write (Port PortNo , PinID PinNo, Pin PinStatus);

Name	**DIO Write**	
API Type	Setter	
Arguments	Port	PortNo
	State which port is used	
	PinID	PinNo
	State which pin is used	
	Pin	PinStatus
	State the value of Pin (High or Low)	
Return Type	void	
Description	responsible for write on the pin for output.	

5. Timer:

API Arguments:

NAME	TIMER_CONFIG
TYPE	Structure
RANGE	Uint8
DESCRIPTION	Configure all timer parameters needed to initialize. It passes a pointer to structure to function for initialization.

API Functions:

Void Timer_Init (Timer_Config * ConfigStruct);

Name	Timer Init	
API Type	Init	
Arguments	Timer_Config *	ConfigStruct
Return Type	void	
Description	Initialize timer with suitable frequency.	

void Timer_Handler (void);

Name	Timer Handler
API Type	Getter
Arguments	void
Return Type	void
Description	ISR fired after reaches specific target determined by application to make MCU attention for certain event.

6. CAN:

API Arguments:

NAME	CAN_CONFIGTYPE
TYPE	Structure
RANGE	Uint8
DESCRIPTION	Contain all configuration used to initialize the CAN correctly. A pointer to structure is passed to the function with all information it needs.

NAME	CANSTATUS
TYPE	Enum
RANGE	0 for PdFalse
	1 for PdTrue
DESCRIPTION	Return the status of the data sent.

API Functions:

void CAN_Init (CAN_Config * ConfigStruct);

Name	**CAN Init**	
API Type	Init	
Arguments	CAN_Config *	ConfigStruct
	Determine the whole data needed to initialize the CAN.	
Return Type	void	
Description	initialize the CAN protocol for communication.	

CANStatus CAN_SendData (uint32 Data);

Name	**CAN SendData**	
API Type	Setter	
Arguments	Uint32	Data
	Contain data needed to be sent via CAN	
Return Type	CANStatus	
Description	responsible for encoding, send data and check if it’s completely sent.	

Uint32 CAN_ReceiveData (void);

Name	**CAN ReceiveData**
API Type	Getter
Arguments	void
Return Type	Uint32
Description	responsible for encoding, send data and check if it’s completely sent.