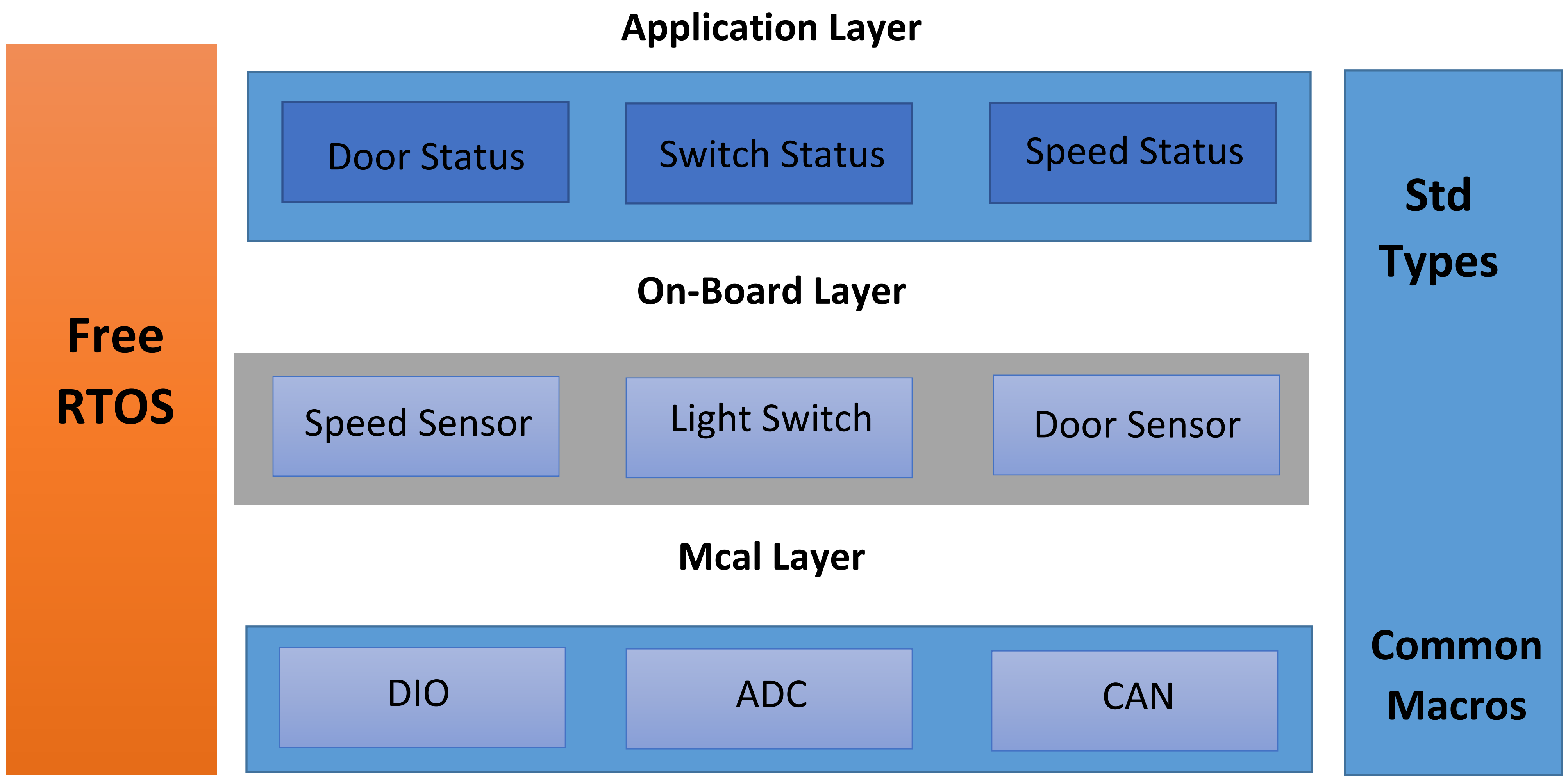
Automotive Door Control System Design (ECU1)



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

For application layer, it contains 3 main tasks:

1. Door Status, which check the status of the door every 10 ms (opened or closed) and send to ECU 2.
2. Switch Status, which check the status of the switch every 20 ms (pressed or released) and send to ECU 2.
3. Speed Status, which check the status of the car every 5 ms (moved or stopped) and send to ECU 2.

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

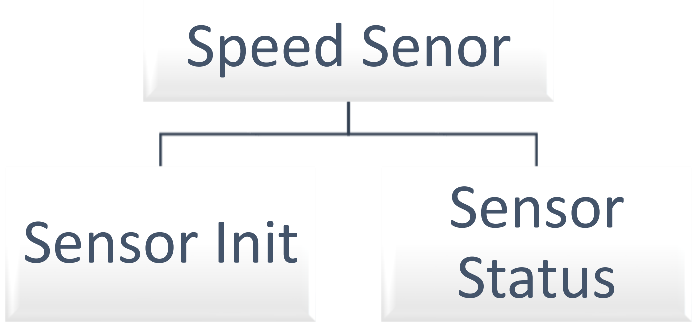
1. Speed Sensor, which measure the speed of the car if it moves or not.
2. Light Switch, which is the output of the action according to the pressed switch or not.
3. Door Sensor, which check the door if its opened or closed for safety.

For Mcal layer, it contains 3 main tasks:

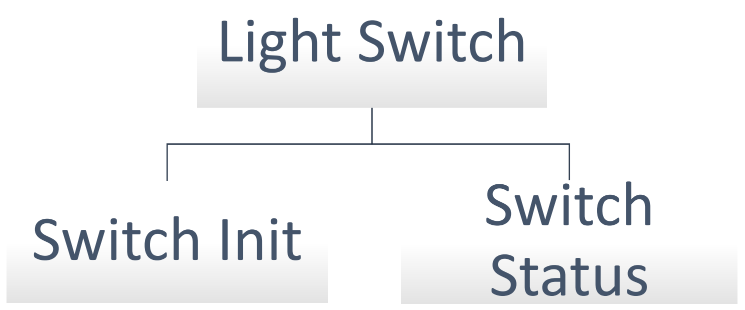
1. ADC, which responsible for converting the analog signal to digital for microcontroller.
2. DIO, which responsible for logic output and input (switch and door status).
3. CAN, which responsible for communicating with another ECU and give the status of the components.

For APIs, that will be used in the Projects:

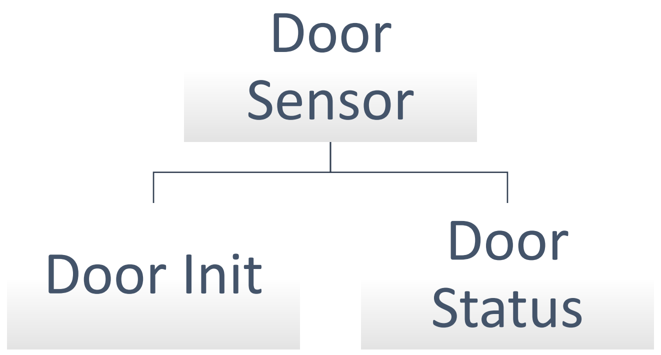
1. Speed Sensor:



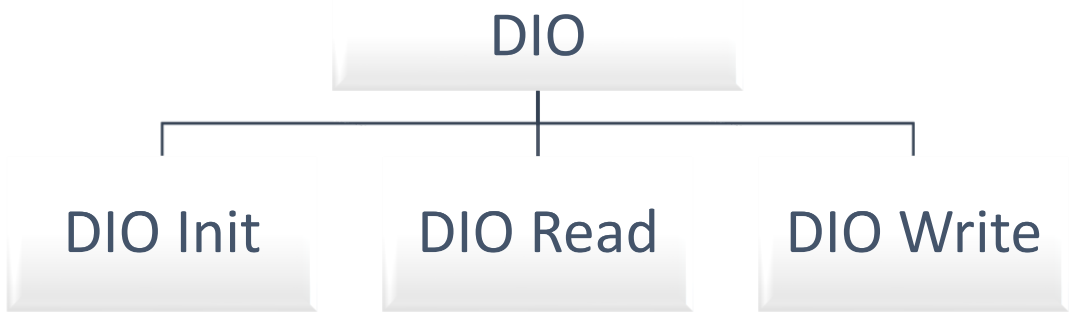
2. Light Switch:



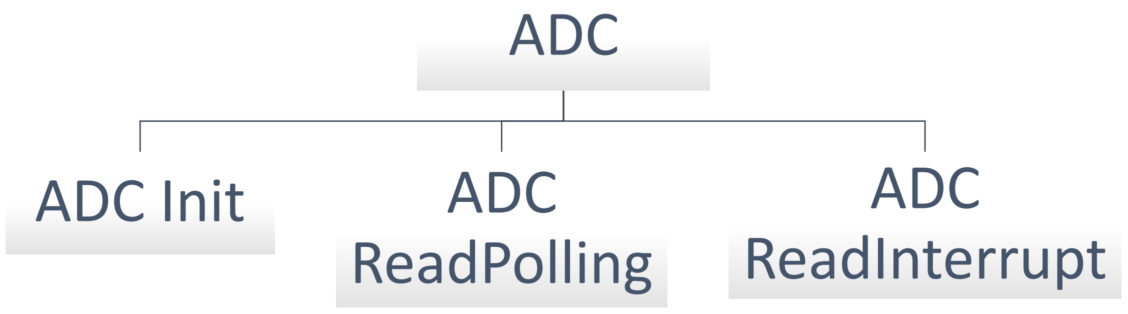
3. Door Sensor:



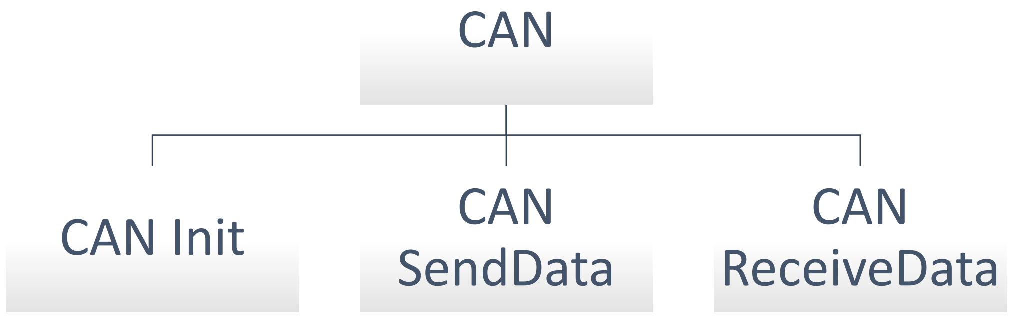
4. DIO:



5. ADC:



6. CAN:



APIs Fully Detailed Description:

1. Speed Sensor:

Must include ADC Driver.

API Arguments:

NAME	**SPEEDSENSOR**
TYPE	Enum
RANGE	0 for SpeedSensor_Off
	1 for SpeedSensor_On
DESCRIPTION	Describe if the pin is high or low.

API Functions:

void Speed_Init (void)

Name	**Speed Init**
API Type	Init
Arguments	void
Return Type	void
Description	that responsible for initialization of the sensor.

SpeedSensor Speed_Status (uint8 Ch-Num);

Name	**Speed Status**	
API Type	Getter	
Arguments	Uint8	Ch-Num
	Select the channel of connected sensor	
Return Type	Speed Sensor	
Description	detect the variable voltage that obtained by the sensor.	

2. Light Switch:

Must include DIO Driver.

API Arguments:

NAME	**LIGHTSWITCH**
TYPE	Enum
RANGE	0 for LightSwitch_Off
	1 for LightSwitch_On
DESCRIPTION	Describe if the pin is high or low.

API Functions:

void Switch_Init (void);

Name	Switch Init
API Type	Init
Arguments	void
Return Type	void
Description	that responsible for initialization of the switch.

LightSwitch Switch_Status (Port PortNo , PinID PinNo, Pin PinStatus);

Name	**Switch 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	LightSwitch	
Description	Detect if switch is pressed or released	

3. Door Sensor:

Must include DIO Driver.

API Arguments:

NAME	DOORSENSOR
TYPE	Enum
RANGE	0 for DoorSensor_Off
	1 for DoorSensor_On
DESCRIPTION	Describe if the pin is high or low.

API Functions:

void Door_Init (void);

Name	Door Init
API Type	Init
Arguments	void
Return Type	void
Description	that responsible for initialization of the sensor.

DoorSensor Door_Status (Port PortNo , PinID PinNo, Pin PinStatus);

Name	**Door 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	DoorSensor	
Description	detect the door if its closed or opened.	

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. ADC:

API Arguments:

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

NAME	**ADC_PRESCALER**
TYPE	Uint8
RANGE	
DESCRIPTION	Define the prescaler used for ADC to work Properly.

NAME	**ADC_REFVOLTAGE**
TYPE	Uint8
RANGE	
DESCRIPTION	Define the reference voltage to set the resolution of the ADC.

API Functions:

void ADC_Init (ADC_ConfigType * ConfigStruct);

Name	**ADC Init**	
API Type	Init	
Arguments	ADC_ConfigType *	ConfigStruct
	Determine the whole data needed to initialize the ADC.	
Return Type	void	
Description	: initialize the ADC with suitable ref. voltage and prescaler	

Uint32 ADC_ReadPolling (uint8 Ch-Num);

Name	**ADC ReadPolling**	
API Type	Getter	
Arguments	Uint8	Ch-Num
	State which channel needed to get data from	
Return Type	Uint32	
Description	determine which channel to read from and polling until get the result of the conversion.	

Void ADC_ReadInterrupt (uint8 Ch-Num);

Name	**ADC ReadInterrupt**	
API Type	Getter	
Arguments	Uint8	Ch-Num
	State which channel needed to get data from	
Return Type	void	
Description	determine which channel to read from and get the data from ISR.	

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