**MOHAMED SAYED HUSSEIN**

Embedded Software Engineer

 E-mail: [*msh.comm@gmail.com*](mailto:msh.comm@gmail.com)

call-512 Cellular: Serbia **+381 628332617 \_** Egypt **+20 1113313898**

 LinkedIn: [*https://www.linkedin.com/in/mohamed-sayed-eng/*](https://www.linkedin.com/in/mohamed-sayed-eng/)

**On-demand Traffic light control for Udacity**

**Static Architecture Design**

1. **Define system layers:**

* Microcontroller Abstraction Layer (MCAL)
* Electronic Unit Abstraction Layer (ECUAL)
* Application Layer

1. **Define system drivers:**

* DIO driver
* Timer driver
* Interrupt driver
* LED driver
* Button driver

1. **Place each driver into the appropriate layer in the appropriate order:**

* MCAL : DIO driver, General purpose TIMER driver, External Interrupt driver
* ECUAL : LED module, Button module
* App Layer : Application module

1. **Define APIs for each driver:**

Define APIS with its documentation, description, input arguments, output arguments, and return:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DIO Function Definition \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Service Name : DIO\_Init

\* Parameters (in) : portNumber, pinNumber, direction, alternative

\* Parameters (out) : None

\* Return value : EN\_returnError\_t

\* Description : Function to Setup the pin configuration:

\* Setup the pin as Digital GPIO pin

\* Setup the internal resistor for i/p pin

\* Setup the mode of alternative function

\* Setup the initial value of the pin High or Low

\* If the input port number or pin number are not correct, The function will return WRONG\_RETURN as EN\_returnError\_t type.

\*/

EN\_returnError\_t **DIO\_Init**(ST\_DIO\_ConfigPin\_t \*pinPTR);

/\*

\* Service Name : DIO\_PortInit

\* Parameters (in) : portNumber, direction

\* Parameters (out) : None

\* Return value : EN\_returnError\_t

\* Description : Function to Setup the PORT configuration:

\* Setup the PORT as Digital GPIO pin

\* Setup the initial value of the port High or Low

\* If the input port number is not correct, The function will return WRONG\_RETURN as EN\_returnError\_t type.

\*/

EN\_returnError\_t **DIO\_PortInit**(uint8 portNumber, EN\_portDirection direction);

/\*

\* Service Name : Dio\_ReadPin

\* Parameters (in) : \*pinPTRr

\* Parameters (out) : \*readValue

\* Return value : EN\_returnError\_t

\* Description : write the value for the required pin, it should be LOGIC\_HIGH or LOGIC\_LOW.

\* If the input port number or pin number are not correct, The function will return WRONG\_RETURN as EN\_returnError\_t type.

\*/

EN\_returnError\_t **Dio\_ReadPin**(ST\_DIO\_ConfigPin\_t \*pinPTR, boolean \*readValue);

/\*

\* Service Name : Dio\_WritePin

\* Parameters (in) : \*pinPTRr, writeValue

\* Parameters (out) : None

\* Return value : EN\_returnError\_t

\* Description : Write input value for the required pin, it should be LOGIC\_HIGH or LOGIC\_LOW.

\* If the input port number or pin number are not correct, The function will return WRONG\_RETURN as EN\_returnError\_t type.

\*/

EN\_returnError\_t **Dio\_WritePin**(ST\_DIO\_ConfigPin\_t \*pinPTR, boolean writeValue);

/\*

\* Service Name : Dio\_ToggelPin

\* Parameters (in) : \*pinPTRr

\* Parameters (out) : None

\* Return value : EN\_returnError\_t

\* Description : Toggle the value for the required pin, it should flapping LOGIC\_HIGH to LOGIC\_LOW and vice versa .

\* If the input port number or pin number are not correct, The function will return WRONG\_RETURN as EN\_returnError\_t type.

\*/

EN\_returnError\_t **Dio\_TogglePin**(ST\_DIO\_ConfigPin\_t \*pinPTR);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* gpTimer Function Definitions \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

* - Frequency = 1MHz
* - Prescaler = 1024
* - Resolution = 1/ (Frequency/ Prescaler) = 1024µs
* - Ticks = 244
* - T = Resolution\*Ticks = 0.25S
* - So you need 4 times timer over flow to get 1 second
* - Timer mode = compare mode
* - Timer initial value = 0
* - Timer compare value = 244

/\*

\* Service Name : Timer\_init

\* Parameters (in) : \*Timer\_Config

\* Parameters (out) : None

\* Return value : EN\_TimerError\_t

\* Description : Function to Setup the TIMER configuration:

\* 1- The Timer Id

\* 2- The Timer Mode (Normal , Compare)

\* 3- The Timer Prescalar

\* 4- The Timer Initial Value That will start counting from it

\* 5- The The Timer Compare Value (In Compare Mode Only)

\* If the Timer initialization does not done correctly,

\* The function will return EN\_TimerError\_t type according to the timer.

\*/

EN\_TimerError\_t **Timer\_init**(**const** ST\_Timer\_Config\_t \*Timer\_ConfigPTR);

/\*

\* Description: Function to set the Call Back function address of Timer0.

\*/

**void** **Timer0\_setCallBack**(**void**(\*T0\_ptr)(**void**));

/\*

\* Description: Function to set the Call Back function address of Timer1.

\*/

**void** **Timer1\_setCallBack**(**void**(\*T1\_ptr)(**void**));

/\*

\* Description: Function to set the Call Back function address of Timer2.

\*/

**void** **Timer2\_setCallBack**(**void**(\*T2\_ptr)(**void**));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* External Interrupt Function Definitions \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Service Name : Ext\_Interrupts\_Init

\* Parameters (in) : \*Ext\_Int\_ConfigPTR

\* Parameters (out) : None

\* Return value : EN\_ExtIntError\_t

\* Description : Function to Setup the External Interrupts configuration:

\* 1- The External Interrupt Id

\* 2- The External Interrupt Mode (LowLevel, ChangeLevel, FailingEdge, RisingEdge)

\* 3- The External Interrupt Enable must be TRUE to Enable INT bit in GICR (General Interrupt Control Register)

\* 4- Enable General Interrupt bit (I-bit)

\* If the External Interrupt initialization does not done correctly,

\* The function will return EN\_ExtIntError\_t type according to the error.

\*/

EN\_ExtIntError\_t **Ext\_Interrupts\_Init**(ST\_Ext\_Int\_Config\_t \*Ext\_Int\_ConfigPTR);

/\*

\* Service Name : Ext\_Interrupts\_deinit

\* Parameters (in) : \*Ext\_Int\_ConfigPTR

\* Parameters (out) : None

\* Return value : EN\_ExtIntError\_t

\* Description : Function to uninitialize the External Interrupts configuration:

\* 1- The External Interrupt Id

\* 2- The External Interrupt must be disabled (Ext\_Int\_Enable = FALSE or 0) in Ext\_Int\_ConfigPTR

\* to clear INT0, INT1, or INT2 bit in GICR (General Interrupt Control Register)

\* If the External Interrupt uninitialized correctly,

\* The function will return EN\_ExtIntError\_t type according to the error.

\*/

EN\_ExtIntError\_t **Ext\_Interrupts\_deinit**(ST\_Ext\_Int\_Config\_t \*Ext\_Int\_ConfigPTR);

/\*

\* Description: Function to set the Call Back function address of External interrupt 0.

\*/

**void** **Ext\_Interrupt0\_setCallBack**(**void**(\*T0\_ptr)(**void**));

/\*

\* Description: Function to set the Call Back function address of External interrupt 1.

\*/

**void** **Ext\_Interrupt1\_setCallBack**(**void**(\*T1\_ptr)(**void**));

/\*

\* Description: Function to set the Call Back function address of External interrupt 2.

\*/

**void** **Ext\_Interrupt2\_setCallBack**(**void**(\*T2\_ptr)(**void**));

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* LED module Function Definitions \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Service Name : LED\_init

\* Parameters (in) : None

\* Parameters (out) : None

\* Return value : EN\_LEDError\_t

\* Description : Function to Setup the LED configuration:

\* If the input port number or pin number are not correct, The function will return WRONG\_RETURN as EN\_LEDError\_t type.

\*/

EN\_LEDError\_t **LED\_init**(**void**);

/\*

\* Service Name : LED\_setOn

\* Parameters (in) : ledPTR

\* Parameters (out) : None

\* Return value : EN\_LEDError\_t

\* Description : Set the LED state to ON

\* in case of fail to set LED to ON, The function will return ERROR\_LED

\*/

EN\_LEDError\_t **LED\_setOn**(ST\_DIO\_ConfigPin\_t \*ledPTR);

/\*

\* Service Name : LED\_setOff

\* Parameters (in) : \*ledPTR

\* Parameters (out) : None

\* Return value : EN\_LEDError\_t

\* Description : Set the LED state to OFF

\* in case of fail to set LED to OFF, The function will return ERROR\_LED

\*/

EN\_LEDError\_t **LED\_setOff**(ST\_DIO\_ConfigPin\_t \*ledPTR);

/\*

\* Service Name : LED\_toggle

\* Parameters (in) : \*ledPTR

\* Parameters (out) : None

\* Return value : EN\_LEDError\_t

\* Description : Toggle the LED state

\* in case of fail to toggling LED state, The function will return ERROR\_LED

\*/

EN\_LEDError\_t **LED\_toggle**(ST\_DIO\_ConfigPin\_t \*ledPTR);

/\*

\* Service Name : LED\_toggle

\* Parameters (in) : \*ledPTR

\* Parameters (out) : None

\* Return value : EN\_LEDError\_t

\* Description : Refresh the LED state, by reading led output value on pin and rewrite the same value

\* in case of fail to Refreshing LED state, The function will return ERROR\_LED

\*/

/\* Description: Refresh the LED state \*/

EN\_LEDError\_t **LED\_refreshOutput**(ST\_DIO\_ConfigPin\_t \*ledPTR);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Button module Function Definitions \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Service Name : Button\_Init

\* Parameters (in) : None

\* Parameters (out) : None

\* Return value : EN\_LEDError\_t

\* Description : Function to Setup the LED configuration:

\* If the led initialization does not done correctly, The function will return EN\_buttonError\_t type according to the DIO or external

\* interrupt error otherwise return OK\_BUTTON

\*/

EN\_buttonError\_t **Button\_Init**(**void**);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Application module Function Definitions \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Service Name : normalMode

\* Parameters (in) : None

\* Parameters (out) : None

\* Return value : EN\_appError\_t

\* Description : Function to call the yellow, red, and green tasks in the normal sequence required

\* If all tasks processed correctly, The function will return OK\_APP

\* Else return as EN\_appError\_t type for error handling according to task fail

\*/

EN\_appError\_t **normalMode**(**void**);

/\*

\* Service Name : pedestrianMode

\* Parameters (in) : None

\* Parameters (out) : None

\* Return value : EN\_appError\_t

\* Description : Function to be called if the Button pressed

\* If all tasks checking the cases of pressing Button correctly, the function will return OK\_APP

\* Else return as EN\_appError\_t type for error handling according to task fail

\*/

EN\_appError\_t **pedestrianMode**(**void**);

/\*

\* Service Name : APP\_Init

\* Parameters (in) : None

\* Parameters (out) : None

\* Return value : EN\_appError\_t

\* Description : Function to Setup the application configuration

\* initialize LEDs as output and Button as input

\* set the call back functions

\* If the initialization processed correctly, the function will return OK\_APP

\* Else return as EN\_appError\_t type for error handling according to task fail

\*/

EN\_appError\_t **APP\_Init**(**void**);

/\*

\* Service Name : APP\_start

\* Parameters (in) : None

\* Parameters (out) : None

\* Return value : EN\_appError\_t

\* Description : Function to start the application by calling modes

\* If mode called correctly, the function will return OK\_APP

\* Else return as EN\_appError\_t type for error handling according to mode fail

\*/

EN\_appError\_t **APP\_start**(**void**);

1. **Define the new data types you will use in these drivers:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DIO driver Data Types \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Enum type to define the pin direction \*/

**typedef** **enum** EN\_pinDirection{

*PIN\_INPUT*=0 , *PIN\_OUTPUT*=1

}EN\_pinDirection;

/\* Enum type to define the port direction \*/

**typedef** **enum** EN\_portDirection{

*PORT\_INPUT*=0 , *PORT\_OUTPUT*=0xFF

}EN\_portDirection;

/\* Enum type to define the alternative function of pin \*/

**typedef** **enum** EN\_pinAlternative{

*PIN\_DIO*=0, *PIN\_ANALOUGE*, *PIN\_COUNTER*, *PIN\_COMPARATOR*, *PIN\_ICU*, *PIN\_SPI\_MOSI*, *PIN\_SPI\_MISO*, *PIN\_UART\_TX*, *PIN\_UART\_RX*, *PIN\_PWM*

}EN\_pinAlternative;

/\* Enum type to define the internal resistor status \*/

**typedef** **enum** EN\_pinInternalResisrot{

*OFF*, *PULL\_UP*, *PULL\_DOWN*

}EN\_pinInternalResisrot;

**typedef** **struct** ST\_DIO\_ConfigPin\_t{

uint8 portNumber;

uint8 pinNumber;

EN\_pinDirection pinDirection; /\* PIN\_INPUT or PIN\_OUTPUT \*/

EN\_pinAlternative alternativeFunction; /\* select the pin mode as GPIO or alternative function... \*/

EN\_pinInternalResisrot pinInternalResistor; /\* OFF, PULL\_UP, or PULL\_DOWN \*/

uint8 pinLevelValue; /\* LOGIC\_HIGH or LOGIC\_LOW \*/

}ST\_DIO\_ConfigPin\_t;

/\* Enum type for return type, to handle APIs error \*/

**typedef** **enum** EN\_returnError\_t

{

*OK\_DOI*, *WRONG\_PIN*, *WRONG\_PORT*, *WRONG\_DIRCTION*

}EN\_returnError\_t;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* gpTimer driver Data Types \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* NOTE: Timer0 : 8-bit counter

\* Timer1 : 16-bit counter

\* Timer2 : 8-bit counter \*/

/\* enum data type to choose one of the 3 AVR timers \*/

**typedef** **enum** EN\_Timer\_Number{

*Timer0*, *Timer1*, *Timer2*

}EN\_Timer\_Number;

/\* enum data type to choose AVR Timer Mode \*/

**typedef** **enum** EN\_Timer\_Modes{

*Normal\_Mode*, *Compare\_Mode*

}EN\_Timer\_Modes;

/\* enum data type to choose AVR Timer Prescalar For Timer 0,1 \*/

**typedef** **enum** EN\_Timer\_Prescalar{

*No\_Clock*, *F\_CPU\_CLOCK*, *F\_CPU\_8*, *F\_CPU\_64*, *F\_CPU\_256*, *F\_CPU\_1024*,

*Ext\_CLK\_Falling\_Edge*, *Ext\_CLK\_Rising\_Edge*

}EN\_Timer\_Prescalar;

/\* enum data type to choose AVR Timer Prescalar For Timer2 as it has different prescalar than TIMERS 0,1 \*/

**typedef** **enum** EN\_Timer2\_Prescalar{

*no\_clock*, *f\_cpu\_clock*, *f\_cpu\_8*, *f\_cpu\_32*, *f\_cpu\_64*, *f\_cpu\_128*,

*f\_cpu\_256*, *f\_Cpu\_1024*

}EN\_Timer2\_Prescalar;

/\* Configuration Structure for AVR Timer Driver Which configure:

1- The timer ID we want to use (0,1,2)

2- The Timer driver modes (NormalMode or Compare\_mode)

3- The Timer Prescalar

4- The Timer Initial value

5- The Timer Compare Value

\*/

**typedef** **struct** ST\_Timer\_Config\_t{

EN\_Timer\_Number Timer\_ID;

EN\_Timer\_Modes Timer\_mode ;

EN\_Timer\_Prescalar Prescalar;

EN\_Timer2\_Prescalar timer2\_prescalar;

uint16 Timer\_Initial\_value;

uint16 Timer\_Compare\_value;

}ST\_Timer\_Config\_t;

/\* Enum type for return type, to handle APIs error \*/

**typedef** **enum** EN\_TimerError\_t{

*ERROR\_TIMER0* ,*ERROR\_TIMER1*, *ERROR\_TIMER2*, *ERROR\_TIMER\_ID*, *ERROR\_TIMER\_MODE*, *OK\_TIMER*

}EN\_TimerError\_t;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* External Interrupt driver Data Types \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Enum data type to choose one of the 3 AVR external interrupt \*/

**typedef** **enum** EN\_Ext\_Interrupt\_Number\_t{

*EXT\_INT0*, *EXT\_INT1*, *EXT\_INT2*

}EN\_Ext\_Interrupt\_Number\_t;

/\* enum data type to choose AVR Timer Mode \*/

**typedef** **enum** EN\_Ext\_Int\_Modes{

*LowLevel*, *HighLevel*, *ChangeLevel*, *FailingEdge*, *RisingEdge*

}EN\_Ext\_Int\_Modes;

/\* Configuration Structure for AVR external interrupt Driver Which configure:

1- The Ext\_Int\_ID we want to use (0,1,2)

2- The Ext\_Int\_Mode driver modes (Low Level, High Level, any Level Change, Failing Edge, or Rising Edge)

3- The Ext\_Int\_Enable want to Enable or Disable (TRUE or FALSE)

\*/

**typedef** **struct** ST\_Ext\_Int\_Config\_t{

EN\_Ext\_Interrupt\_Number\_t Ext\_Int\_ID;

EN\_Ext\_Int\_Modes Ext\_Int\_Mode;

boolean Ext\_Int\_Enable;

}ST\_Ext\_Int\_Config\_t;

/\* Enum type for return type, to handle APIs error \*/

**typedef** **enum** EN\_ExtIntError\_t{

*ERROR\_EXT\_INT0* ,*ERROR\_EXT\_INT1*, *ERROR\_EXT\_INT2*,

*ERROR\_EXT\_INT\_ID*, *ERROR\_EXT\_INT\_MODE*, *OK\_EXT\_INT*

}EN\_ExtIntError\_t;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* LED module Data Types \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Set the led ON/OFF according to its configuration Positive logic or negative logic \*/

**#define** LED\_ON LOGIC\_HIGH

**#define** LED\_OFF LOGIC\_LOW

/\* Enum type for return type, to handle APIs error \*/

**typedef** **enum** EN\_LEDError\_t{

*OK\_LED*, *ERROR\_LED* ,*ERROR\_LED\_RED*, *ERROR\_LED\_YALLOW*, *ERROR\_LED\_GREEN*

}EN\_LEDError\_t;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Button module Data Types \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Enum type for return type, to handle APIs error \*/

**typedef** **enum** EN\_buttonError\_t{

*OK\_BUTTON*, *ERROR\_BUTTON\_DIO\_INIT*, *ERROR\_BUTTON\_EXT\_INT\_INIT*

}EN\_buttonError\_t;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Application Module Data Types \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* Enum type for return type, to handle APIs error \*/

**typedef** **enum** EN\_appError\_t{

*OK\_APP*, *ERROR\_APP\_LED\_INIT*, *ERROR\_APP\_BUTTON\_INIT*, *ERROR\_APP\_TIMER\_INIT*,

*ERROR\_APP\_RED\_ON*, *ERROR\_APP\_YELLOW\_ON*, *ERROR\_APP\_GREEN\_ON*,

*ERROR\_NORMAL\_MODE*, *ERROR\_PEDESTRIAN\_MODE*

}EN\_appError\_t;