SYSTEM DESIGN

On-demand Traffic light control

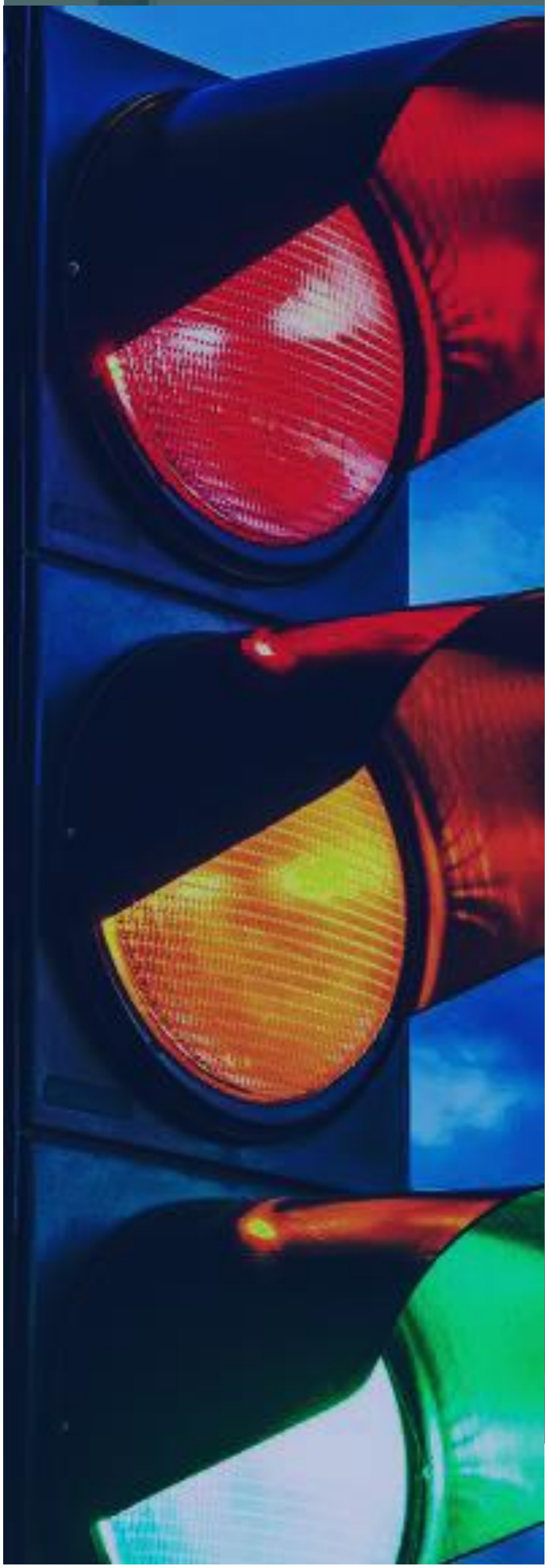


TABLE OF CONTENTS

System Description:

This system works in two specific modes:

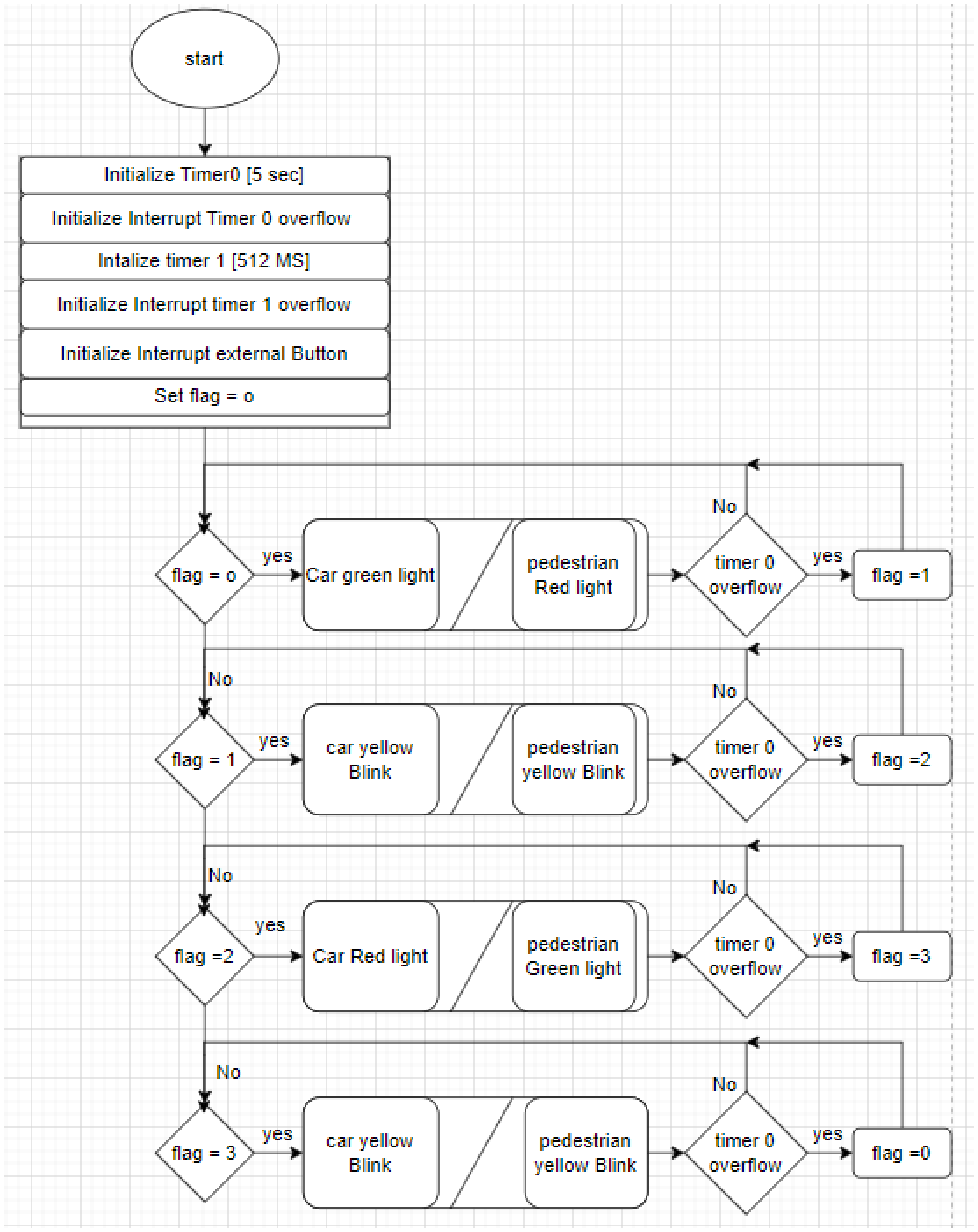
In normal mode:

1. Cars' LEDs will be changed every five seconds starting from Green then yellow then red then yellow then Green.
2. The Yellow LED will blink for five seconds before moving to Green or Red LEDs.

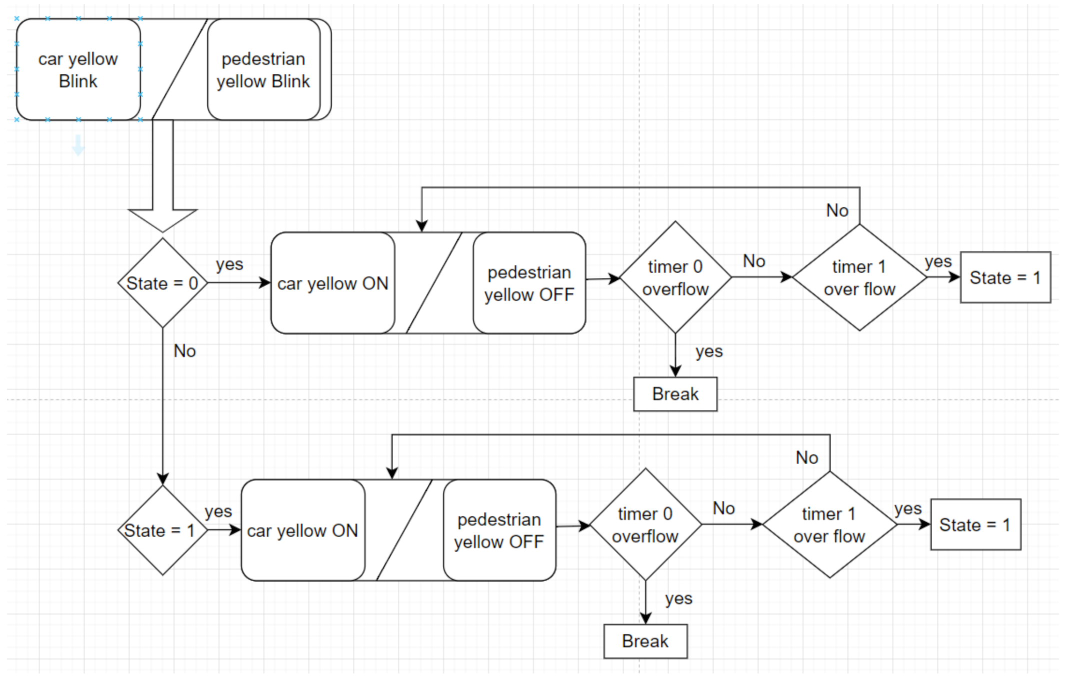
In pedestrian mode:

1. Change from normal mode to pedestrian mode when the pedestrian button is pressed.
2. If pressed when the cars' Red LED is on, the pedestrian's Green LED and the cars' Red LEDs will be on for five seconds, this means that pedestrians can cross the street while the pedestrian's Green LED is on.
3. If pressed when the cars' Green LED is on or the cars' Yellow LED is blinking, the pedestrian Red LED will be on then both Yellow LEDs start to blink for five seconds, then the cars' Red LED and pedestrian Green LEDs are on for five seconds, this means that pedestrian must wait until the Green LED is on.
4. At the end of the two states, the cars' Red LED will be off and both Yellow LEDs start blinking for 5 seconds and the pedestrian's Green LED is still on.
5. After the five seconds the pedestrian Green LED will be off and both the pedestrian Red LED and the cars' Green LED will be on.
6. Traffic lights signals are going to the normal mode again.

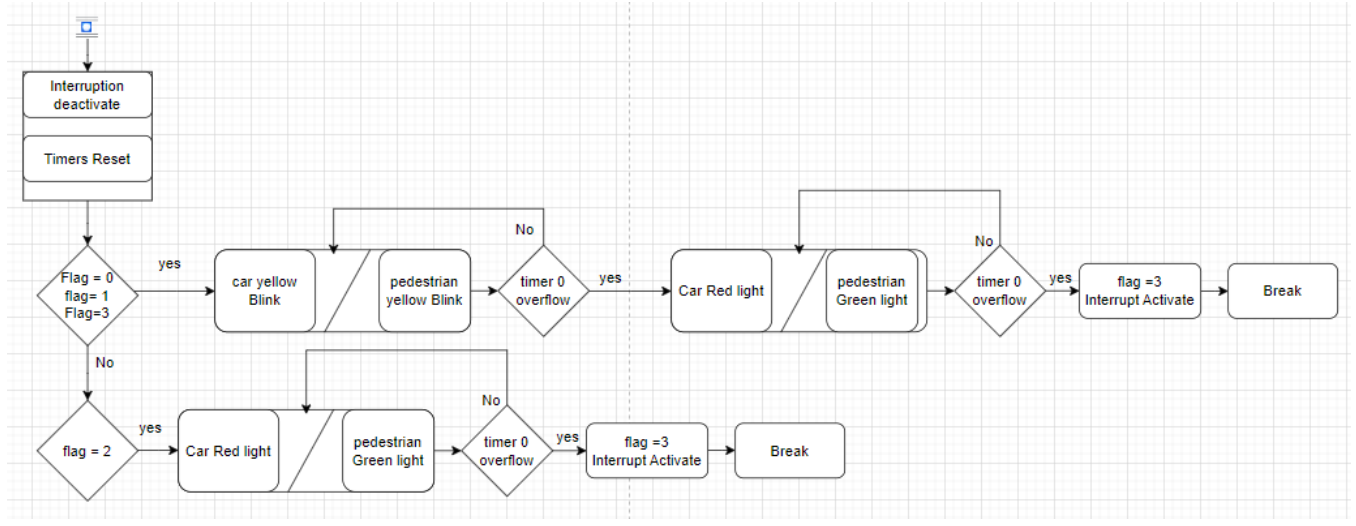
Flow Chart: Main Program



Flow Chart: Yellow Blinking



Flow Chart: Button Interrupt



System layers:

This system is divided into 4 layers:

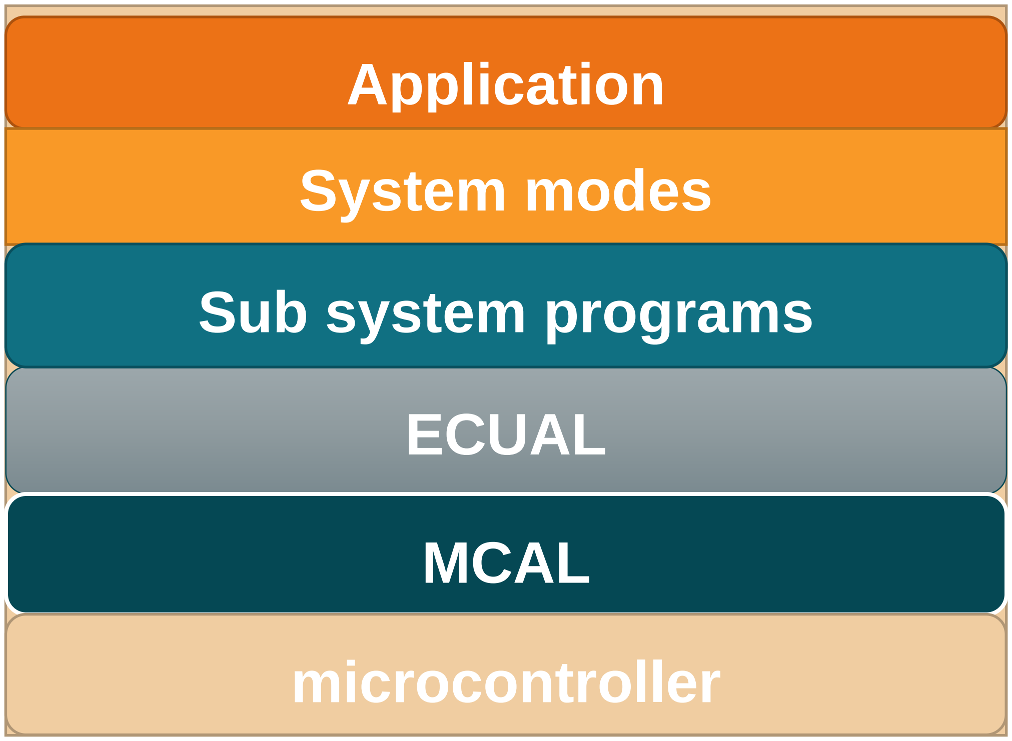
- Microcontroller
- MCAL
- ECUAL
- Sub system modes
- System mods
- Application

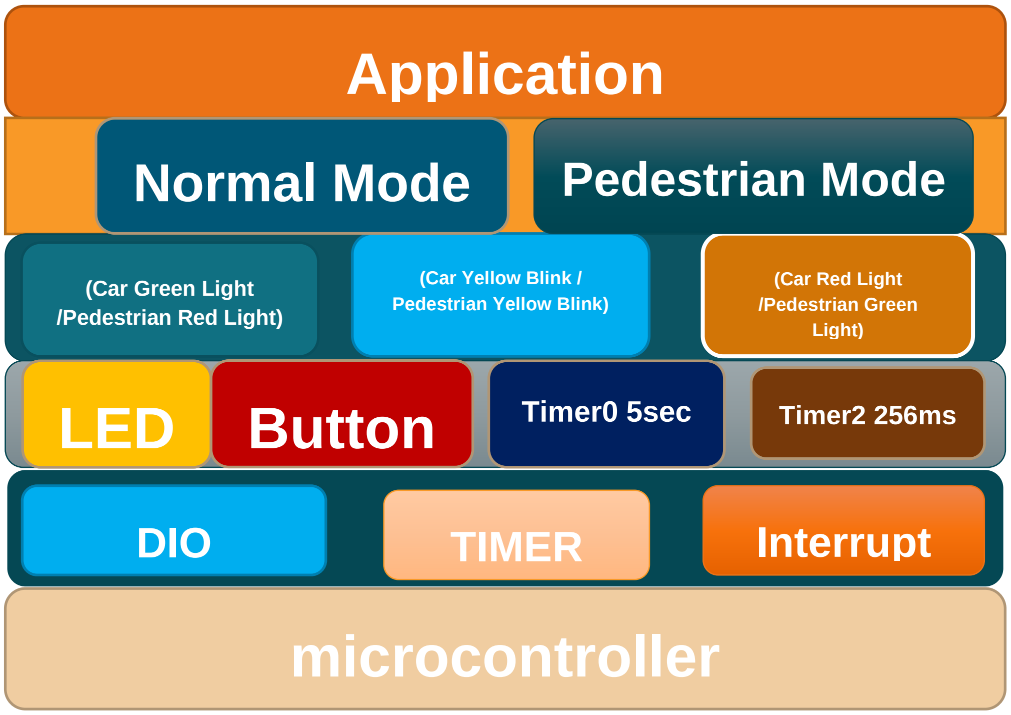
System Drivers:

The system is composed of 12 drivers:

- Dio
- Timer
- Interrupt
- Button
- Led
- Timer 0 (5sec)
- Timer 2 (256ms)
- (Car Green Light /Pedestrian Red Light)
- (Car Yellow Blink / Pedestrian Yellow Blink)
- (Car Red Light /Pedestrian Green Light)
- Normal Mode
- Pedestrian Mode
- Application

Drivers In Their Appropriate Layer:

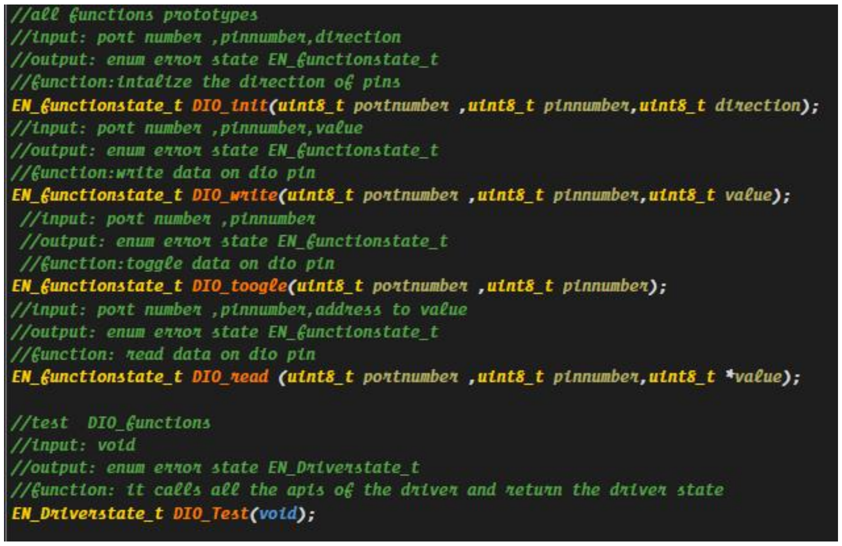



Application
Normal Mode
Pedestrian Mode
(Car Green Light /Pedestrian Red Light)
(Car Yellow Blink / Pedestrian Yellow Blink)
(Car Red Light /Pedestrian Green Light)
LED
Button
Timer0 5sec
Timer2 256ms
DIO
TIMER
Interrupt
microcontroller

Drivers APIS:

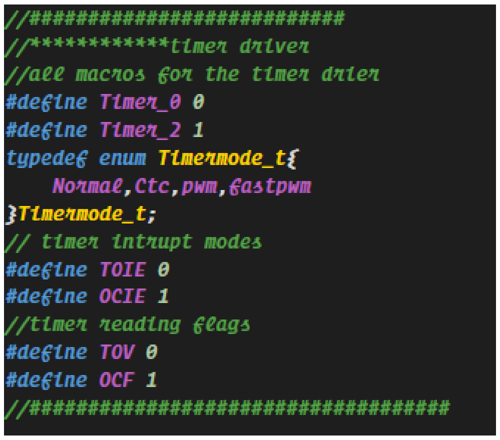
I.MCAL.APIS

- ### DIO

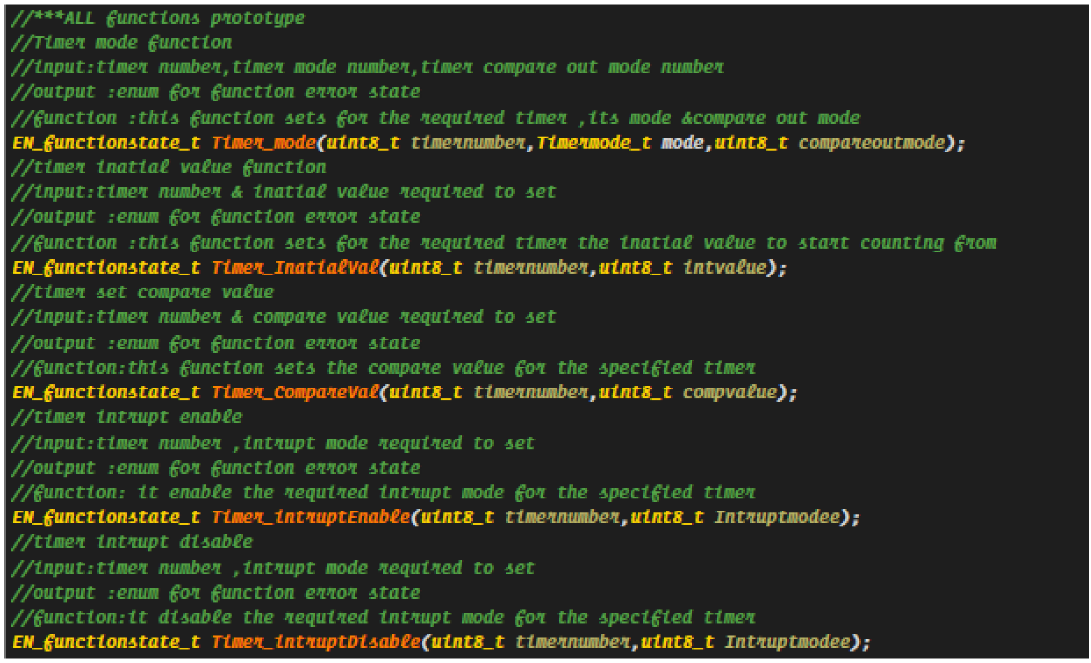


```
//all functions prototypes
//input: port number ,pinnumber,direction
//output: enum error state EN_functionstate_t
//function:intalize the direction of pins
EN_functionstate_t DIO_init(uint8_t portnumber ,uint8_t pinnumber,uint8_t direction);
//input: port number ,pinnumber,value
//output: enum error state EN_functionstate_t
//function:write data on dio pin
EN_functionstate_t DIO_write(uint8_t portnumber ,uint8_t pinnumber,uint8_t value);
 //input: port number ,pinnumber
 //output: enum error state EN_functionstate_t
 //function:toggle data on dio pin
EN_functionstate_t DIO_toogle(uint8_t portnumber ,uint8_t pinnumber);
//input: port number ,pinnumber,address to value
//output: enum error state EN_functionstate_t
//function: read data on dio pin
EN_functionstate_t DIO_read (uint8_t portnumber ,uint8_t pinnumber,uint8_t *value);

//test  DIO_functions
//input: void
//output: enum error state EN_Driverstate_t
//function: it calls all the apis of the driver and return the driver state
EN_Driverstate_t DIO_Test(void);
```

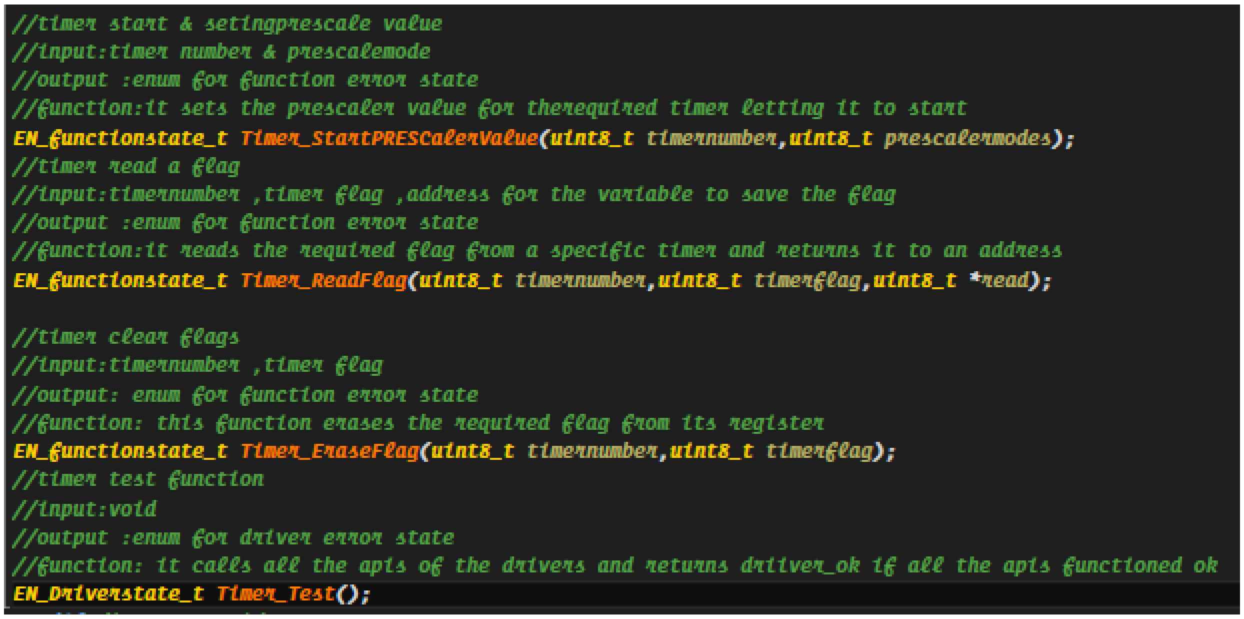
- ### TIMER



```
//###########################
//************timer driver
//all macros for the timer drier
#define Timer_0 0
#define Timer_2 1
typedef enum Timermode_t{
    Normal,Ctc,pwm,fastpwm
}Timermode_t;
// timer intrupt modes
#define TOIE 0
#define OCIE 1
//timer reading flags
#define TOV 0
#define OCF 1
//####################################
```

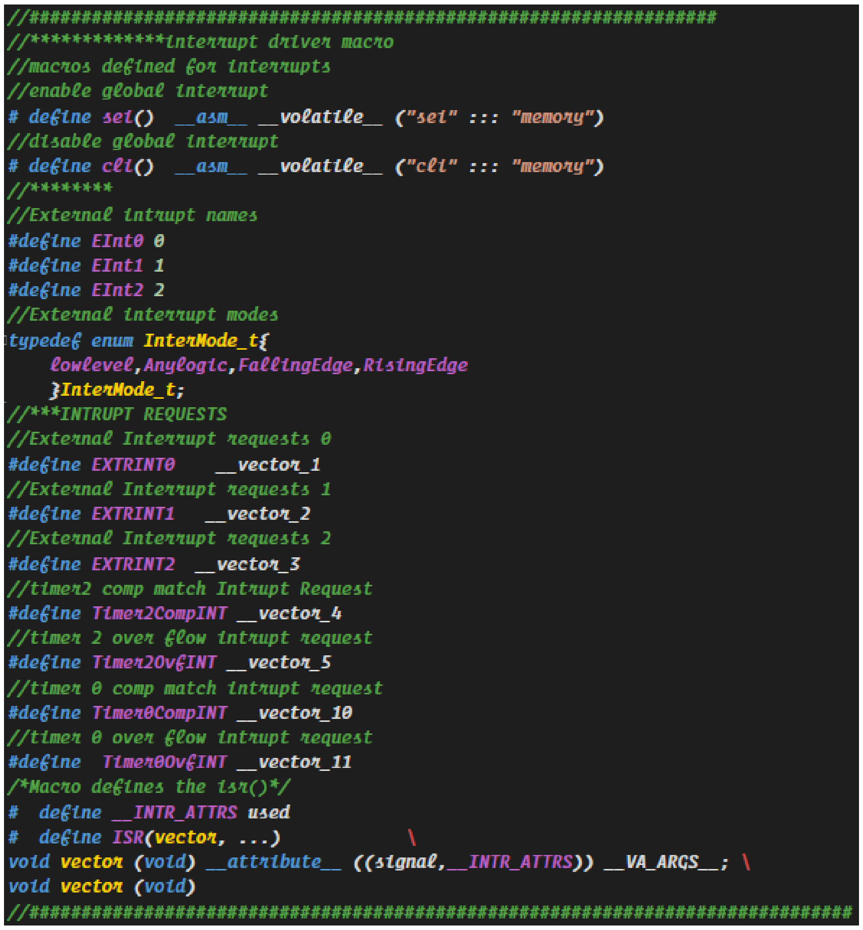


```
//***ALL functions prototype
//Timer mode function
//input:timer number,timer mode number,timer compare out mode number
//output :enum for function error state
//function :this function sets for the required timer ,its mode &compare out mode
EN_functionstate_t Timer_mode(uint8_t timernumber,Timermode_t mode,uint8_t compareoutmode);
//timer inatial value function
//input:timer number & inatial value required to set
//output :enum for function error state
//function :this function sets for the required timer the inatial value to start counting from
EN_functionstate_t Timer_InatialVal(uint8_t timernumber,uint8_t intvalue);
//timer set compare value
//input:timer number & compare value required to set
//output :enum for function error state
//function:this function sets the compare value for the specified timer
EN_functionstate_t Timer_CompareVal(uint8_t timernumber,uint8_t compvalue);
//timer intrupt enable
//input:timer number ,intrupt mode required to set
//output :enum for function error state
//function: it enable the required intrupt mode for the specified timer
EN_functionstate_t Timer_intruptEnable(uint8_t timernumber,uint8_t Intruptmodee);
//timer intrupt disable
//input:timer number ,intrupt mode required to set
//output :enum for function error state
//function:it disable the required intrupt mode for the specified timer
EN_functionstate_t Timer_intruptDisable(uint8_t timernumber,uint8_t Intruptmodee);
```

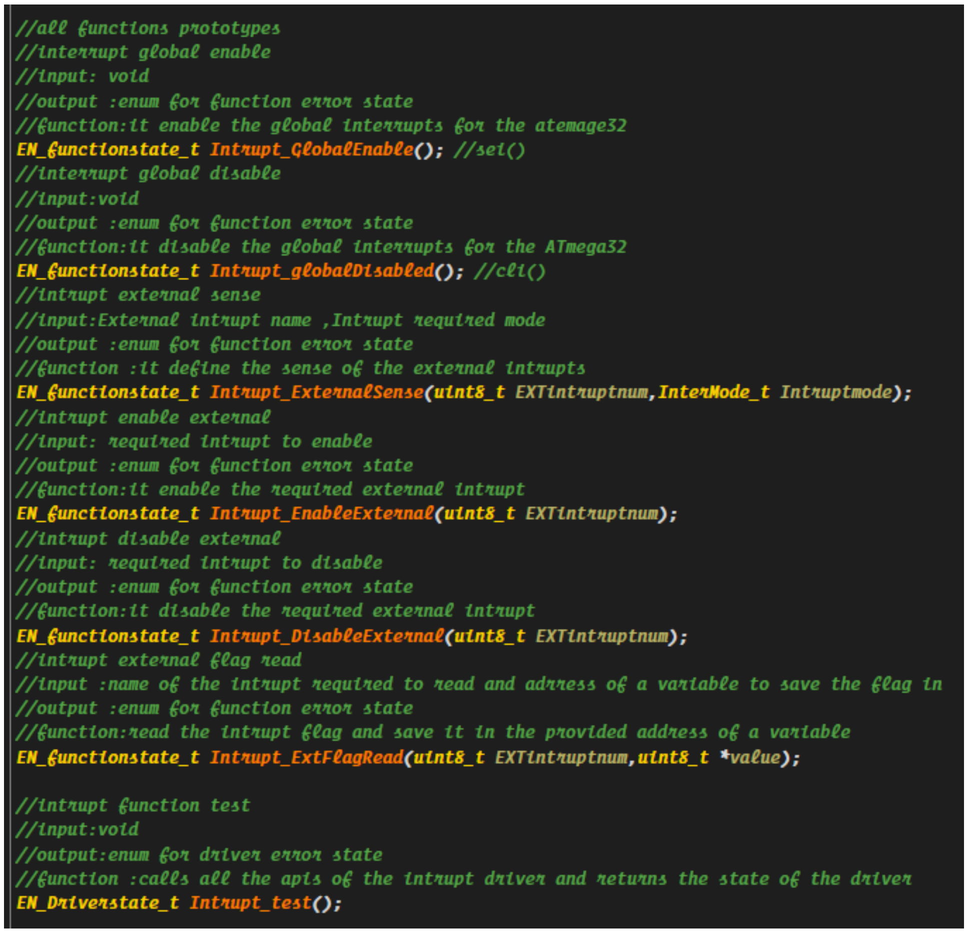


```
//timer start & setingprescale value
//input:timer number & prescalemode
//output :enum for function error state
//function:it sets the prescaler value for therequired timer letting it to start
EN_functionstate_t Timer_StartPREScalerValue(uint8_t timernumber,uint8_t prescalermodes);
//timer read a flag
//input:timernumber ,timer flag ,address for the variable to save the flag
//output :enum for function error state
//function:it reads the required flag from a specific timer and returns it to an address
EN_functionstate_t Timer_ReadFlag(uint8_t timernumber,uint8_t timerflag,uint8_t *read);

//timer clear flags
//input:timernumber ,timer flag
//output: enum for function error state
//function: this function erases the required flag from its register
EN_functionstate_t Timer_EraseFlag(uint8_t timernumber,uint8_t timerflag);
//timer test function
//input:void
//output :enum for driver error state
//function: it calls all the apis of the drivers and returns driver_ok if all the apis functioned ok
EN_Driverstate_t Timer_Test();
```

- **INTERRUPT**



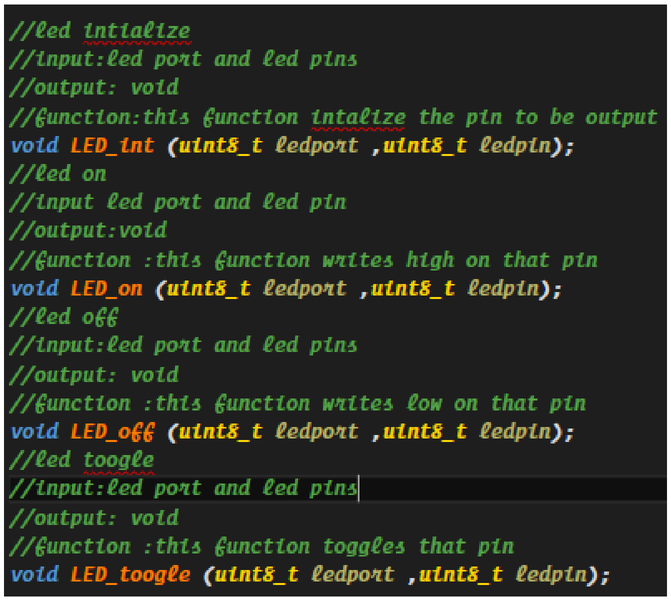
```
//###################################################################
//*************interrupt driver macro
//macros defined for interrupts
//enable global interrupt
# define sei()  __asm__ __volatile__ ("sei" ::: "memory")
//disable global interrupt
# define cli()  __asm__ __volatile__ ("cli" ::: "memory")
//********
//External intrupt names
#define EInt0 0
#define EInt1 1
#define EInt2 2
//External interrupt modes
typedef enum InterMode_t{
    lowlevel,Anylogic,FallingEdge,RisingEdge
    }InterMode_t;
//***INTRUPT REQUESTS
//External Interrupt requests 0
#define EXTRINT0    __vector_1
//External Interrupt requests 1
#define EXTRINT1   __vector_2
//External Interrupt requests 2
#define EXTRINT2  __vector_3
//timer2 comp match Intrupt Request
#define Timer2CompINT __vector_4
//timer 2 over flow intrupt request
#define Timer2OvfINT __vector_5
//timer 0 comp match intrupt request
#define Timer0CompINT __vector_10
//timer 0 over flow intrupt request
#define  Timer0OvfINT __vector_11
/*Macro defines the isr()*/
#  define __INTR_ATTRS used
#  define ISR(vector, ...)            \
void vector (void) __attribute__ ((signal,__INTR_ATTRS)) __VA_ARGS__; \
void vector (void)
//################################################################################
```



```
//all functions prototypes
//interrupt global enable
//input: void
//output :enum for function error state
//function:it enable the global interrupts for the atemage32
EN_functionstate_t Intrupt_GlobalEnable(); //sei()
//interrupt global disable
//input:void
//output :enum for function error state
//function:it disable the global interrupts for the ATmega32
EN_functionstate_t Intrupt_globalDisabled(); //cli()
//intrupt external sense
//input:External intrupt name ,Intrupt required mode
//output :enum for function error state
//function :it define the sense of the external intrupts
EN_functionstate_t Intrupt_ExternalSense(uint8_t EXTintruptnum,InterMode_t Intruptmode);
//intrupt enable external
//input: required intrupt to enable
//output :enum for function error state
//function:it enable the required external intrupt
EN_functionstate_t Intrupt_EnableExternal(uint8_t EXTintruptnum);
//intrupt disable external
//input: required intrupt to disable
//output :enum for function error state
//function:it disable the required external intrupt
EN_functionstate_t Intrupt_DisableExternal(uint8_t EXTintruptnum);
//intrupt external flag read
//input :name of the intrupt required to read and adress of a variable to save the flag in
//output :enum for function error state
//function:read the intrupt flag and save it in the provided address of a variable
EN_functionstate_t Intrupt_ExtFlagRead(uint8_t EXTintruptnum,uint8_t *value);

//intrupt function test
//input:void
//output:enum for driver error state
//function :calls all the apis of the intrupt driver and returns the state of the driver
EN_Driverstate_t Intrupt_test();
```

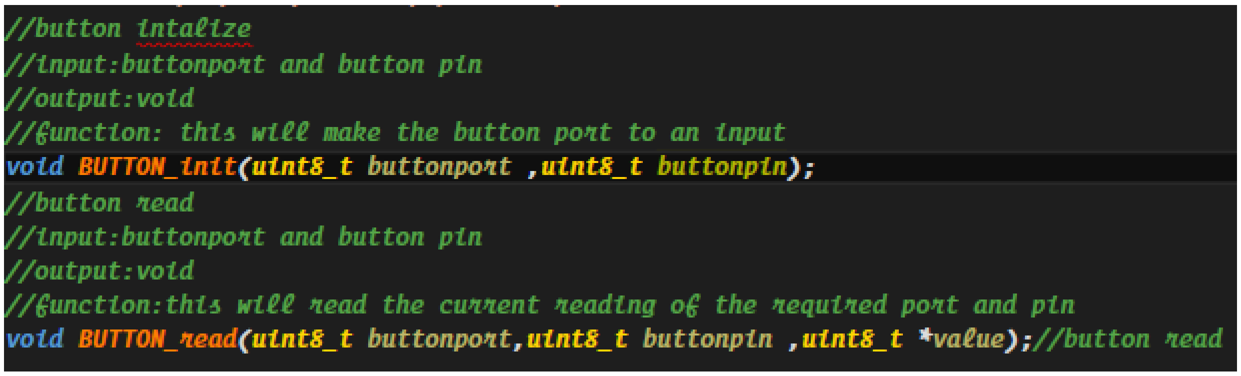
II.ECUAL.APIS

- **LED**



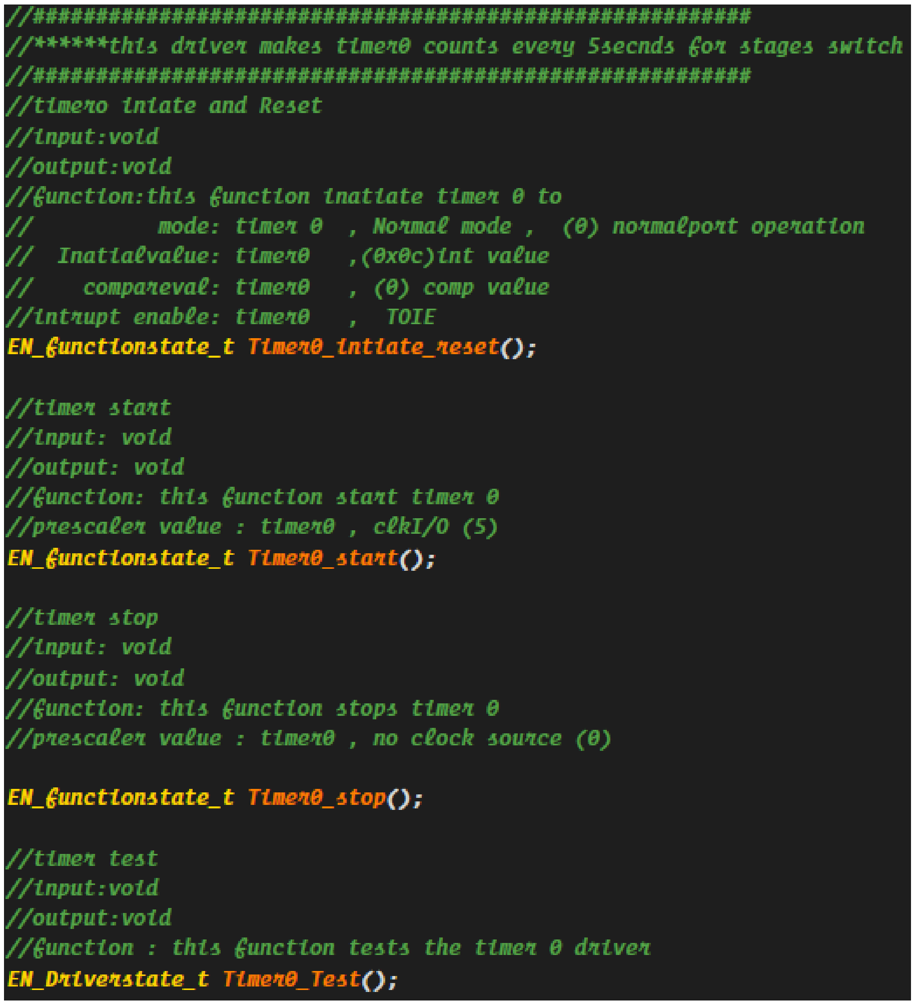
```
//led intialize
//input:led port and led pins
//output: void
//function:this function intalize the pin to be output
void LED_int (uint8_t ledport ,uint8_t ledpin);
//led on
//input led port and led pin
//output:void
//function :this function writes high on that pin
void LED_on (uint8_t ledport ,uint8_t ledpin);
//led off
//input:led port and led pins
//output: void
//function :this function writes low on that pin
void LED_off (uint8_t ledport ,uint8_t ledpin);
//led toogle
//input:led port and led pins
//output: void
//function :this function toggles that pin
void LED_toogle (uint8_t ledport ,uint8_t ledpin);
```

- ## BUTTON



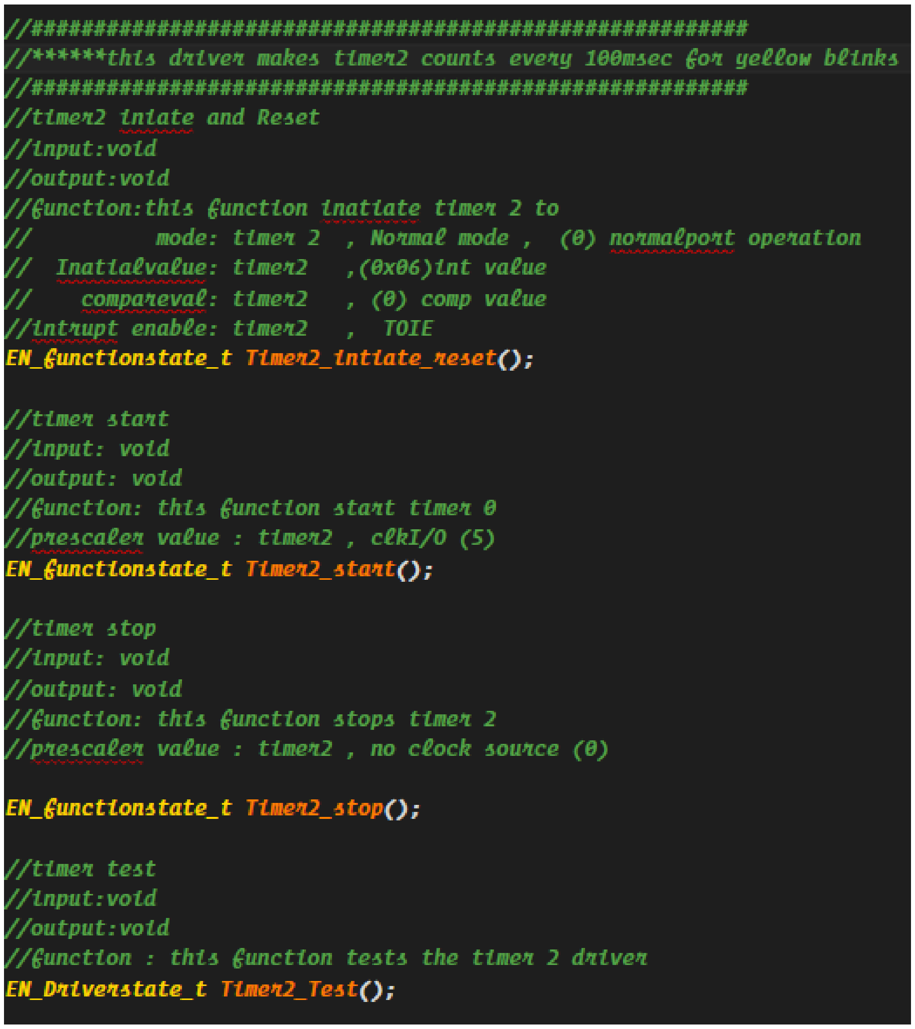
```
//button intalize
//input:buttonport and button pin
//output:void
//function: this will make the button port to an input
void BUTTON_init(uint8_t buttonport ,uint8_t buttonpin);
//button read
//input:buttonport and button pin
//output:void
//function:this will read the current reading of the required port and pin
void BUTTON_read(uint8_t buttonport,uint8_t buttonpin ,uint8_t *value);//button read
```

- ## TIMER0 5SEC



```
//##########################################################
//******this driver makes timer0 counts every 5secnds for stages switch
//##########################################################
//timero intate and Reset
//input:void
//output:void
//function:this function inatiate timer 0 to
//          mode: timer 0   , Normal mode ,  (0) normalport operation
//  Inatialvalue: timer0    ,(0x0c)int value
//    compareval: timer0    , (0) comp value
//intrupt enable: timer0    ,  TOIE
EN_functionstate_t Timer0_intiate_reset();

//timer start
//input: void
//output: void
//function: this function start timer 0
//prescaler value : timer0 , clkI/O (5)
EN_functionstate_t Timer0_start();

//timer stop
//input: void
//output: void
//function: this function stops timer 0
//prescaler value : timer0 , no clock source (0)

EN_functionstate_t Timer0_stop();

//timer test
//input:void
//output:void
//function : this function tests the timer 0 driver
EN_Driverstate_t Timer0_Test();
```

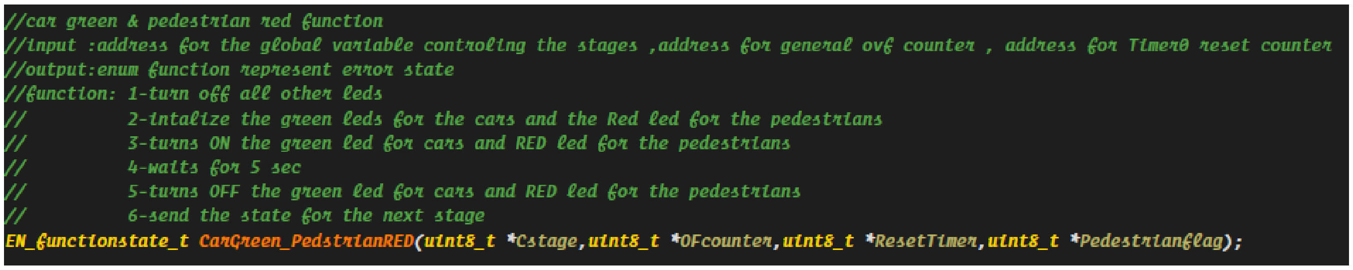
- ## TIMER2 256MSEC



```
//########################################################
//******this driver makes timer2 counts every 100msec for yellow blinks
//########################################################
//timer2 intiate and Reset
//input:void
//output:void
//function:this function inatiate timer 2 to
//          mode: timer 2   , Normal mode ,  (0) normalport operation
//  Inatialvalue: timer2   ,(0x06)int value
//    compareval: timer2   , (0) comp value
//intrupt enable: timer2   ,  TOIE
EN_functionstate_t Timer2_intiate_reset();

//timer start
//input: void
//output: void
//function: this function start timer 0
//prescaler value : timer2 , clkI/O (5)
EN_functionstate_t Timer2_start();

//timer stop
//input: void
//output: void
//function: this function stops timer 2
//prescaler value : timer2 , no clock source (0)

EN_functionstate_t Timer2_stop();

//timer test
//input:void
//output:void
//function : this function tests the timer 2 driver
EN_Driverstate_t Timer2_Test();
```

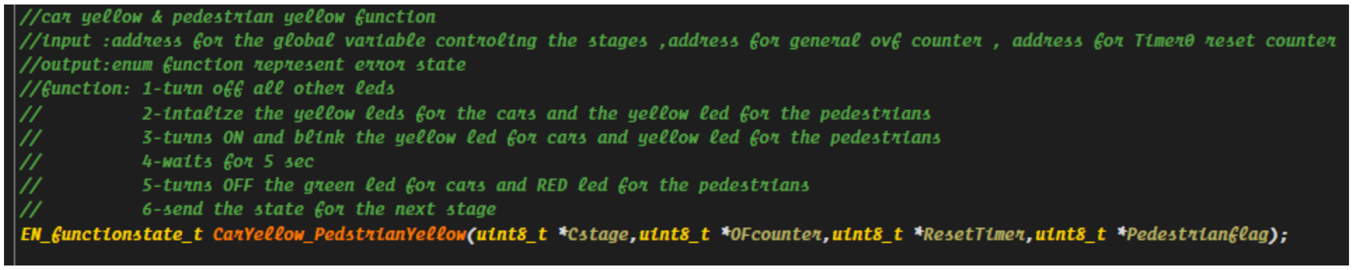
III.SUB SYSTEM MODES.APIS

- ### (CAR GREEN LIGHT /PEDESTRIAN RED LIGHT) [CG_PR]



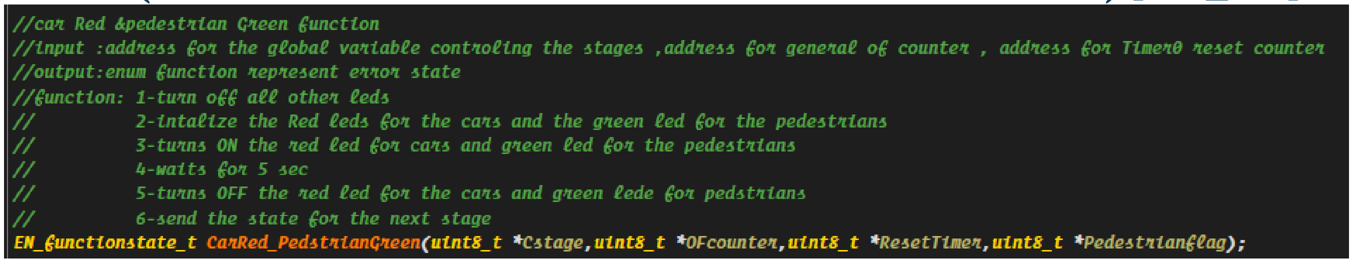
```
//car green & pedestrian red function
//input :address for the global variable controling the stages ,address for general ovf counter , address for Timer0 reset counter
//output:enum function represent error state
//function: 1-turn off all other leds
//          2-intalize the green leds for the cars and the Red led for the pedestrians
//          3-turns ON the green led for cars and RED led for the pedestrians
//          4-waits for 5 sec
//          5-turns OFF the green led for cars and RED led for the pedestrians
//          6-send the state for the next stage
EN_functionstate_t CarGreen_PedstrianRED(uint8_t *Cstage,uint8_t *OFcounter,uint8_t *ResetTimer,uint8_t *Pedestrianflag);
```

- **(CAR YELLOW / PEDESTRIAN YELLOW BLINK) [CY_PY]**



```
//car yellow & pedestrian yellow function
//input :address for the global variable controling the stages ,address for general ovf counter , address for Timer0 reset counter
//output:enum function represent error state
//function: 1-turn off all other leds
//          2-intalize the yellow leds for the cars and the yellow led for the pedestrians
//          3-turns ON and blink the yellow led for cars and yellow led for the pedestrians
//          4-waits for 5 sec
//          5-turns OFF the green led for cars and RED led for the pedestrians
//          6-send the state for the next stage
EN_functionstate_t CarYellow_PedstrianYellow(uint8_t *Cstage,uint8_t *OFcounter,uint8_t *ResetTimer,uint8_t *Pedestrianflag);
```

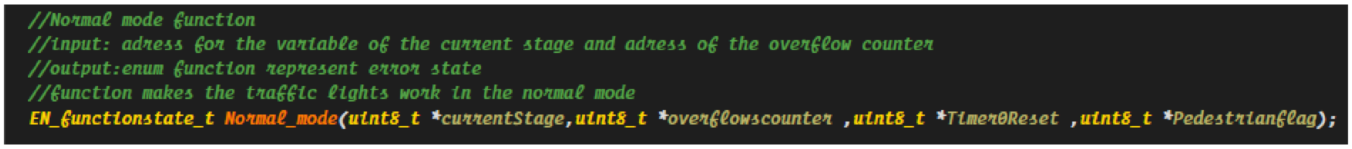
- **(CAR RED LIGHT /PEDESTRIAN GREEN LIGHT) [CR_PG]**



```
//car Red &pedestrian Green function
//input :address for the global variable controling the stages ,address for general of counter , address for Timer0 reset counter
//output:enum function represent error state
//function: 1-turn off all other leds
//          2-intalize the Red leds for the cars and the green led for the pedestrians
//          3-turns ON the red led for cars and green led for the pedestrians
//          4-waits for 5 sec
//          5-turns OFF the red led for the cars and green lede for pedstrians
//          6-send the state for the next stage
EN_functionstate_t CarRed_PedstrianGreen(uint8_t *Cstage,uint8_t *OFcounter,uint8_t *ResetTimer,uint8_t *Pedestrianflag);
```

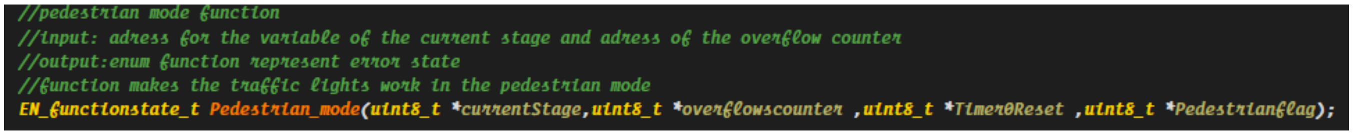
IV.SYSTEM MODES .APIS

- **NORMAL MODE**



```
//Normal mode function
//input: adress for the variable of the current stage and adress of the overflow counter
//output:enum function represent error state
//function makes the traffic lights work in the normal mode
EN_functionstate_t Normal_mode(uint8_t *currentStage,uint8_t *overflowscounter ,uint8_t *Timer0Reset ,uint8_t *Pedestrianflag);
```

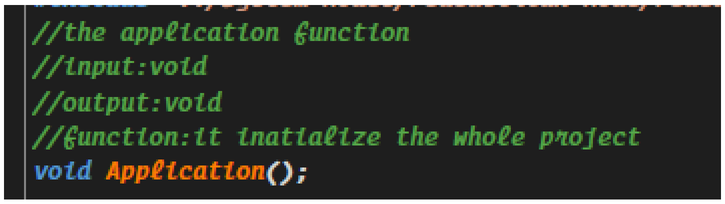
- **PEDESTRIAN MODE**



```
//pedestrian mode function
//input: adress for the variable of the current stage and adress of the overflow counter
//output:enum function represent error state
//function makes the traffic lights work in the pedestrian mode
EN_functionstate_t Pedestrian_mode(uint8_t *currentStage,uint8_t *overflowscounter ,uint8_t *Timer0Reset ,uint8_t *Pedestrianflag);
```

V.APPLICATION.API

- **APPLICATION**



```
//the application function
//input:void
//output:void
//function:it inatialize the whole project
void Application();
```

system constrains

the only external constrain that affects the system is the pedestrian button:

- **pressing the button, a short press:**

  the system will go to the pedestrian mode were it will take action depending on the current state of the traffic light following button interrupt: flow chart

- **pressing the button, a long press:**

  the system will not take any action and will just continue in the Normal mode

- **pressing the button double clicks:**

  the system will only respond to the first press taking the suitable action from the pedestrian mode and will not response for any farther clicks until returning back to the normal mode again