On Demand Traffic Light System

Project report

Name : Mohamed Ashraf Elsayed Mahmoud

Email add. : mohamedashrafelsayed321@gmail.com

August Cohort

System Description:

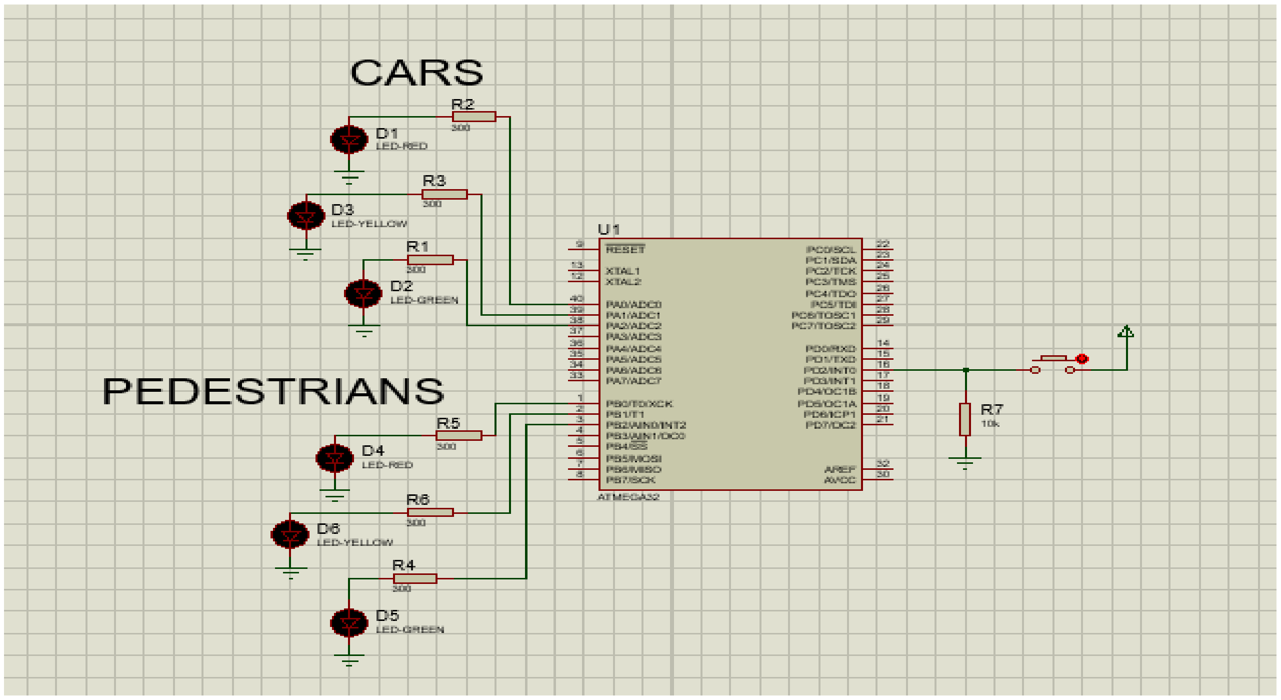
The system is a traffic light with two modes:
normal mode and pedestrian mode. Switching between the two modes is done using a push button. The system is controlled by microcontroller which is ATmega32a.

NORMAL MODE:

In normal mode , when the cars' red light is on the pedestrians' green light is also on . when cars' yellow light blinks pedestrians' yellow lights also blinks . when cars' green light is on the pedestrians' red light is also on .so if we considered cars it would go on a sequence of red/yellow/green/yellow/red…… and the pedestrians are its inverse except for the yellow color that is common between them.

PEDESTRIAN MODE:

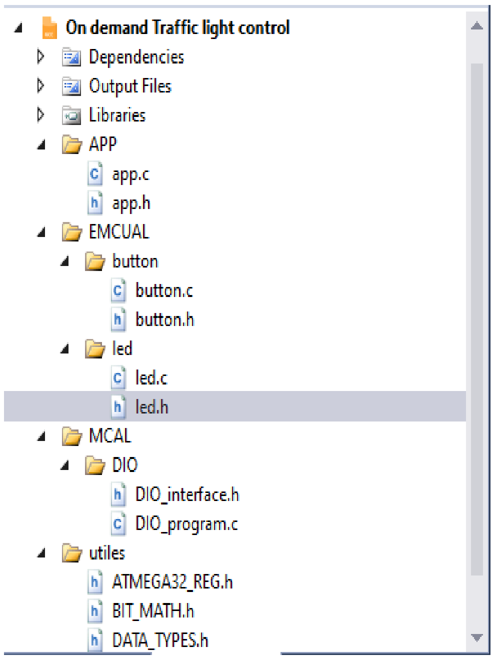
When switching to the pedestrian mode by pressing on the button, the system checks first which pair of light is on. If the cars' red light is on and the pedestrian green light is on too. The system does nothing but if the cars' green or yellow light is on the yellow light on both cars and pedestrians traffic blinks for five seconds then the cars' red light will be on while pedestrians green light will be on.



This is the simulation of the project on proteus

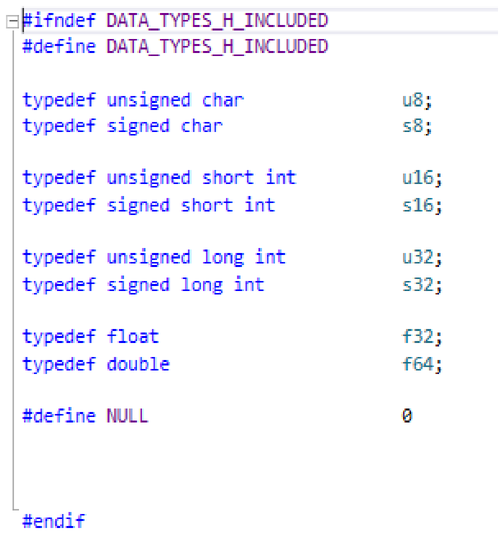
System Design and Implementations:

I used the static architecture and SOLID principles in creating and programming my project so I divided the whole project into layers like MCAL, ECUAL and APP in addition to a folder called utiles that contain 3 files which are (DATATYPES.H , BITMATH.H and ATMEGA32REG.h) that can be used in each and every layer and also can be used in any project concerning the use of atmega32 which is great for reusability.



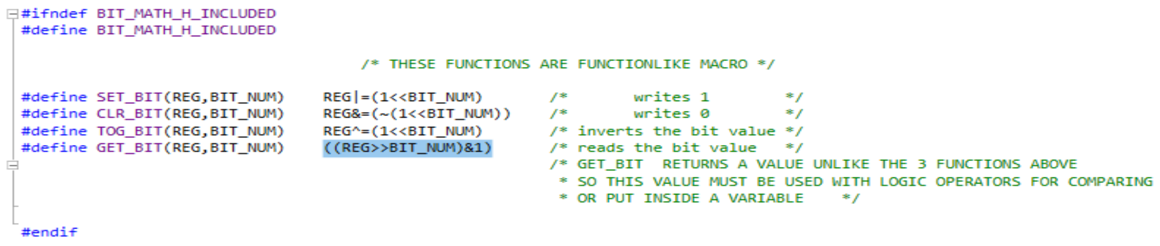
```
On demand Traffic light control
  Dependencies
  Output Files
  Libraries
  APP
    app.c
    app.h
  EMCUAL
    button
      button.c
      button.h
    led
      led.c
      led.h
  MCAL
    DIO
      DIO_interface.h
      DIO_program.c
  utiles
    ATMEGA32_REG.h
    BIT_MATH.h
    DATA_TYPES.h
```

This is the solution explorer that shows the layers ,folders and files.



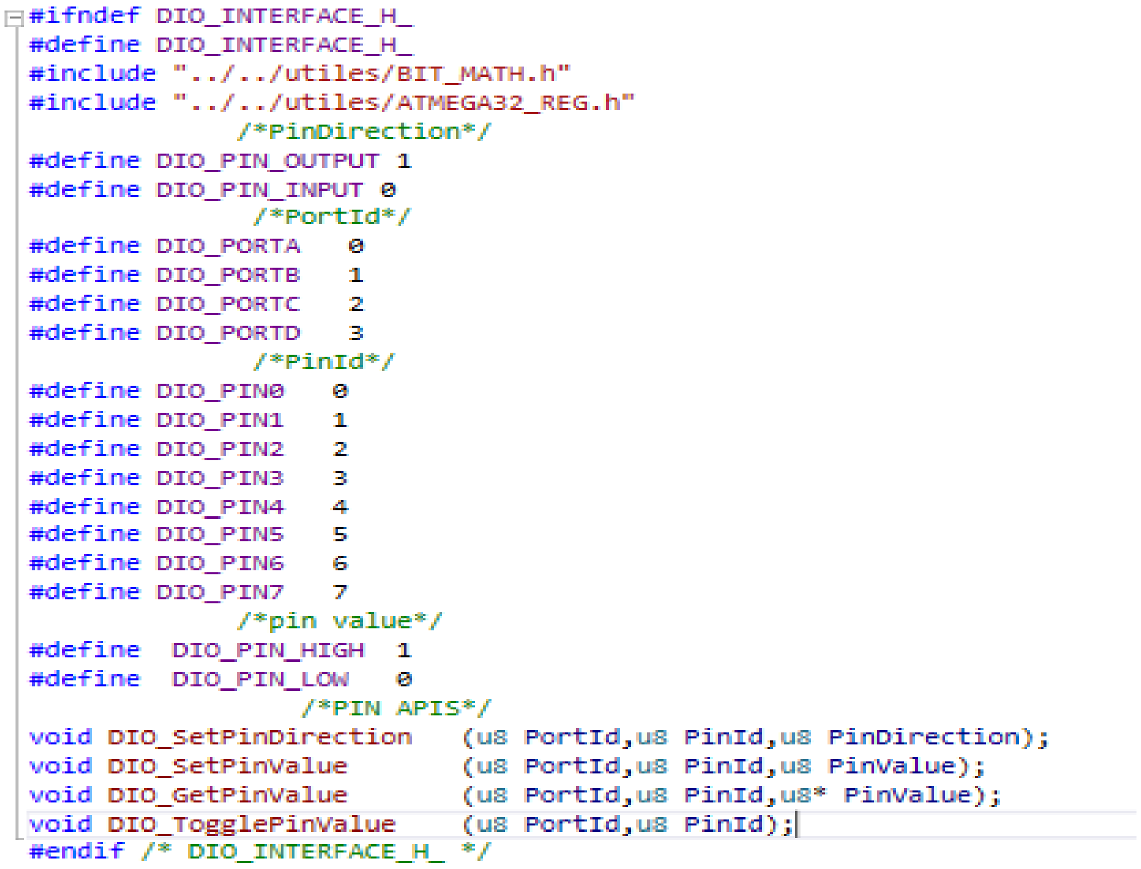
```
#ifndef DATA_TYPES_H_INCLUDED
#define DATA_TYPES_H_INCLUDED

typedef unsigned char                u8;
typedef signed char                  s8;

typedef unsigned short int           u16;
typedef signed short int             s16;

typedef unsigned long int            u32;
typedef signed long int              s32;

typedef float                        f32;
typedef double                       f64;

#define NULL                         0


#endif
```

This is the datatypes file that is inside the utiles folder and here I wrote all the typedefs



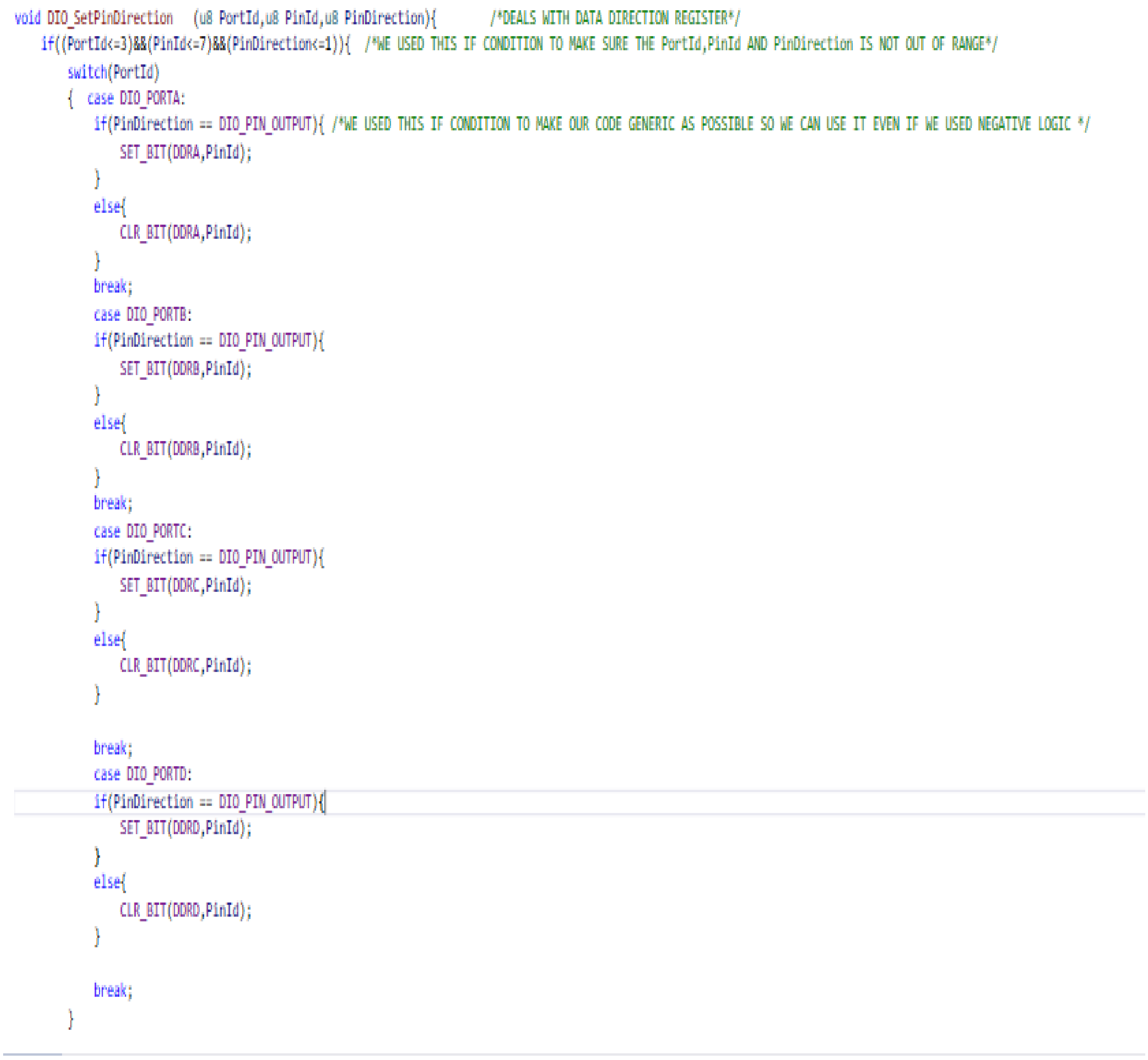
```c
#ifndef BIT_MATH_H_INCLUDED
#define BIT_MATH_H_INCLUDED

                                  /* THESE FUNCTIONS ARE FUNCTIONLIKE MACRO */

#define SET_BIT(REG,BIT_NUM)    REG|=(1<<BIT_NUM)       /*       writes 1        */
#define CLR_BIT(REG,BIT_NUM)    REG&=(~(1<<BIT_NUM))    /*       writes 0        */
#define TOG_BIT(REG,BIT_NUM)    REG^=(1<<BIT_NUM)       /* inverts the bit value */
#define GET_BIT(REG,BIT_NUM)    ((REG>>BIT_NUM)&1)      /* reads the bit value   */
                                                        /* GET_BIT  RETURNS A VALUE UNLIKE THE 3 FUNCTIONS ABOVE
                                                         * SO THIS VALUE MUST BE USED WITH LOGIC OPERATORS FOR COMPARING
                                                         * OR PUT INSIDE A VARIABLE    */

#endif
```

this is BIT_MATH.H file which contains 4 function like macros that set, clear, toggle and read the value of the bit .



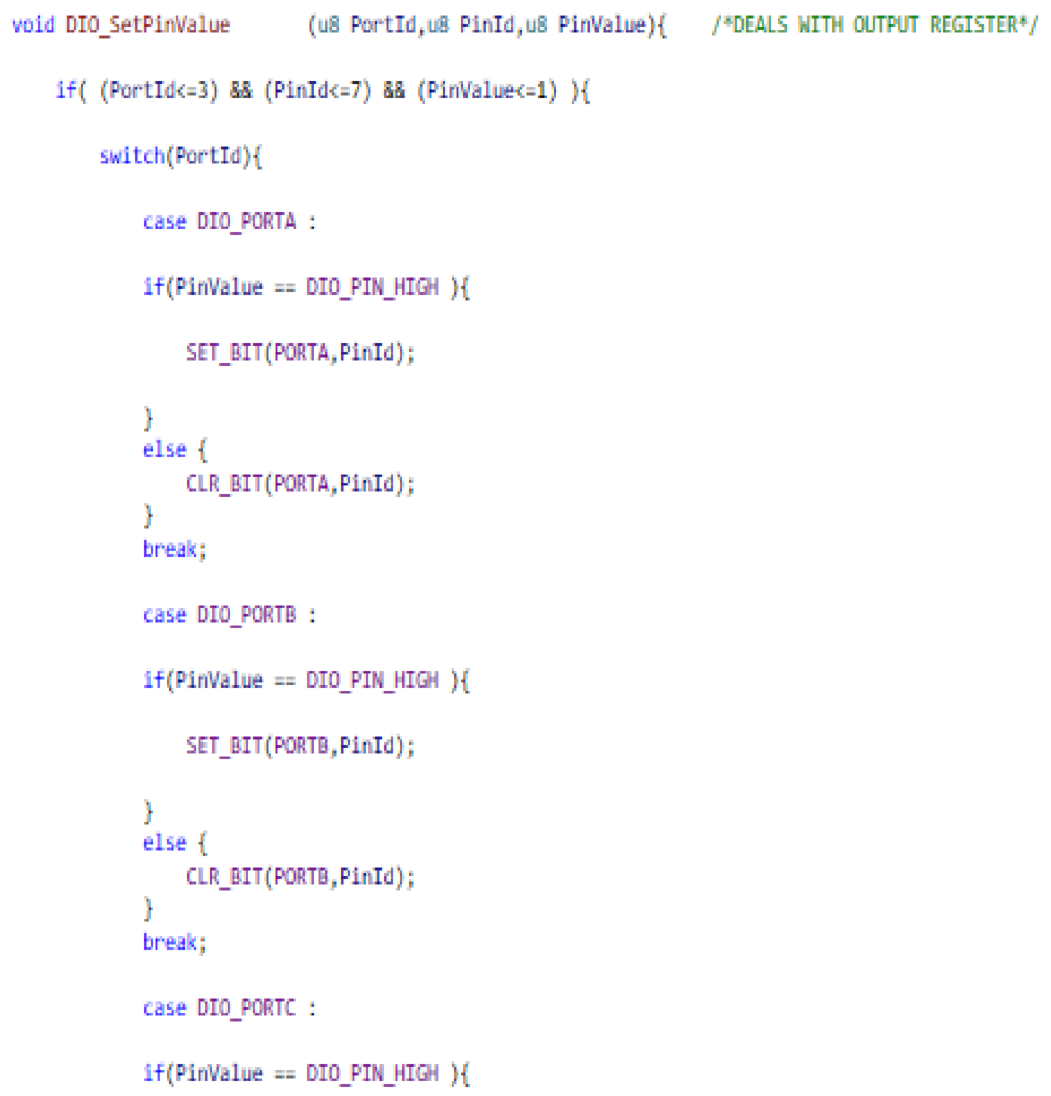
```c
#ifndef DIO_INTERFACE_H_
#define DIO_INTERFACE_H_
#include "../../utiles/BIT_MATH.h"
#include "../../utiles/ATMEGA32_REG.h"
             /*PinDirection*/
#define DIO_PIN_OUTPUT 1
#define DIO_PIN_INPUT 0
              /*PortId*/
#define DIO_PORTA    0
#define DIO_PORTB    1
#define DIO_PORTC    2
#define DIO_PORTD    3
              /*PinId*/
#define DIO_PIN0    0
#define DIO_PIN1    1
#define DIO_PIN2    2
#define DIO_PIN3    3
#define DIO_PIN4    4
#define DIO_PIN5    5
#define DIO_PIN6    6
#define DIO_PIN7    7
             /*pin value*/
#define  DIO_PIN_HIGH   1
#define  DIO_PIN_LOW    0
                 /*PIN APIS*/
void DIO_SetPinDirection   (u8 PortId,u8 PinId,u8 PinDirection);
void DIO_SetPinValue       (u8 PortId,u8 PinId,u8 PinValue);
void DIO_GetPinValue       (u8 PortId,u8 PinId,u8* PinValue);
void DIO_TogglePinValue    (u8 PortId,u8 PinId);
#endif /* DIO_INTERFACE_H_ */
```

this is the dio header file where I defined macros like pin direction, port id, pin id and pin value in addition to the prototypes of 4 functions .



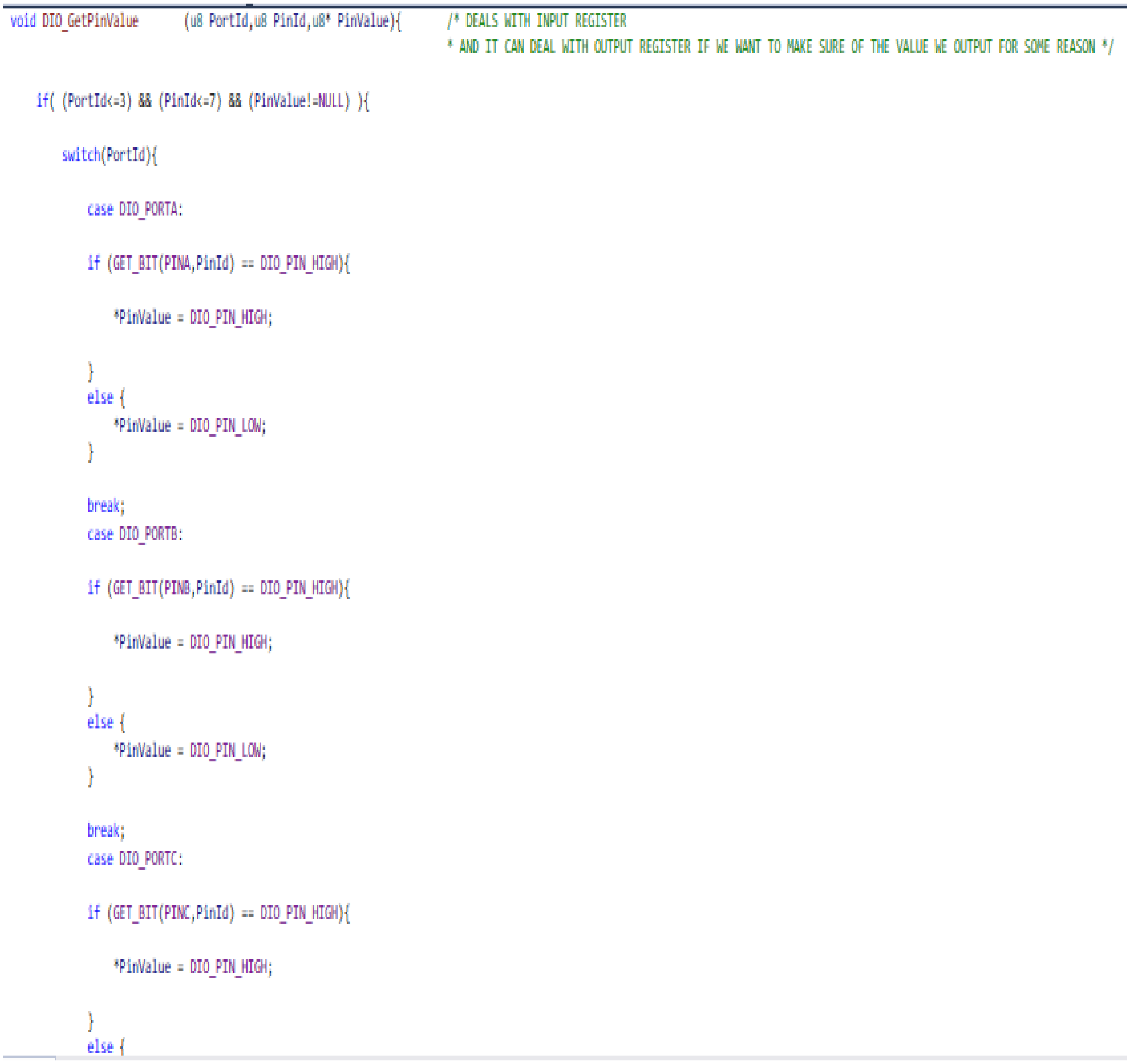
```
void DIO_SetPinDirection   (u8 PortId,u8 PinId,u8 PinDirection){       /*DEALS WITH DATA DIRECTION REGISTER*/
    if((PortId<=3)&&(PinId<=7)&&(PinDirection<=1)){  /*WE USED THIS IF CONDITION TO MAKE SURE THE PortId,PinId AND PinDirection IS NOT OUT OF RANGE*/
        switch(PortId)
        {  case DIO_PORTA:
            if(PinDirection == DIO_PIN_OUTPUT){ /*WE USED THIS IF CONDITION TO MAKE OUR CODE GENERIC AS POSSIBLE SO WE CAN USE IT EVEN IF WE USED NEGATIVE LOGIC */
                SET_BIT(DDRA,PinId);
            }
            else{
                CLR_BIT(DDRA,PinId);
            }
            break;
            case DIO_PORTB:
            if(PinDirection == DIO_PIN_OUTPUT){
                SET_BIT(DDRB,PinId);
            }
            else{
                CLR_BIT(DDRB,PinId);
            }
            break;
            case DIO_PORTC:
            if(PinDirection == DIO_PIN_OUTPUT){
                SET_BIT(DDRC,PinId);
            }
            else{
                CLR_BIT(DDRC,PinId);
            }

            break;
            case DIO_PORTD:
            if(PinDirection == DIO_PIN_OUTPUT){
                SET_BIT(DDRD,PinId);
            }
            else{
                CLR_BIT(DDRD,PinId);
            }

            break;
        }
```

this is the first function in dio.c that sets the direction of the pin whether it is output or input and takes the port, pin and output or input .



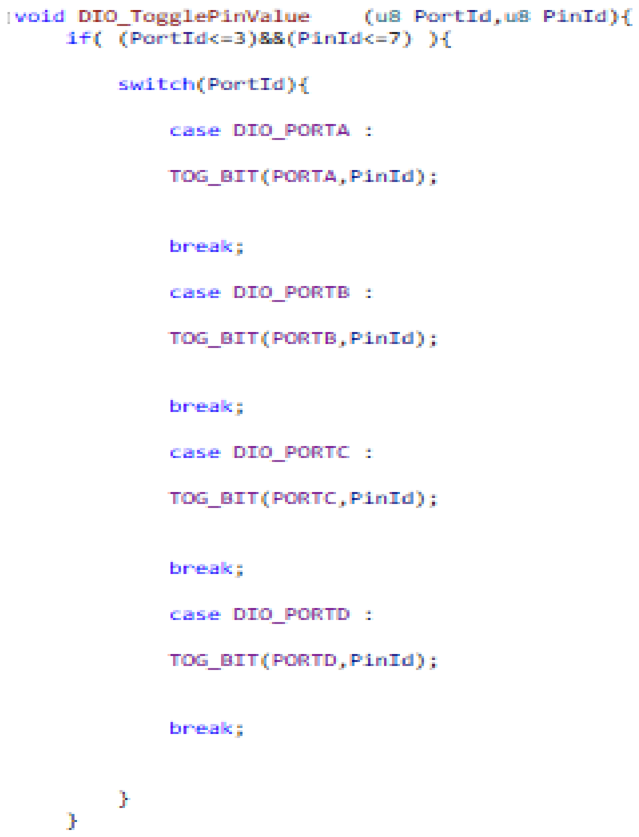
```
void DIO_SetPinValue      (u8 PortId,u8 PinId,u8 PinValue){    /*DEALS WITH OUTPUT REGISTER*/

    if( (PortId<=3) && (PinId<=7) && (PinValue<=1) ){

        switch(PortId){

            case DIO_PORTA :

            if(PinValue == DIO_PIN_HIGH ){

                SET_BIT(PORTA,PinId);

            }
            else {
                CLR_BIT(PORTA,PinId);
            }
            break;

            case DIO_PORTB :

            if(PinValue == DIO_PIN_HIGH ){

                SET_BIT(PORTB,PinId);

            }
            else {
                CLR_BIT(PORTB,PinId);
            }
            break;

            case DIO_PORTC :

            if(PinValue == DIO_PIN_HIGH ){
```

this function sets the value of a bit to high or low according to the value passed to it.



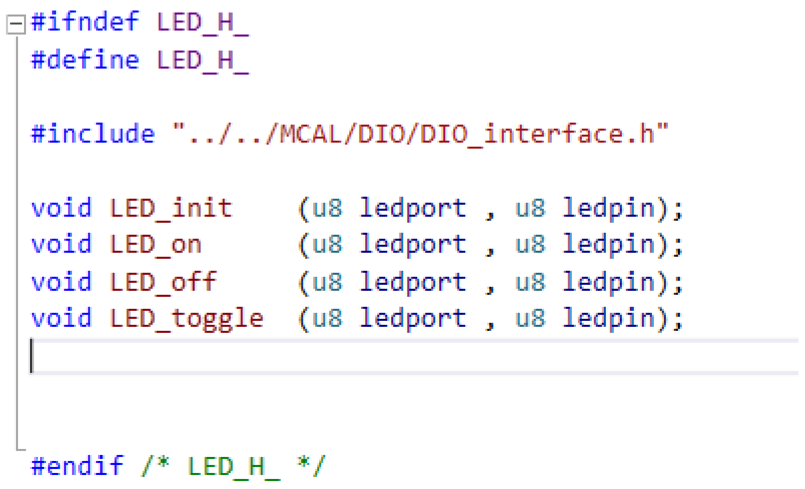
```
void DIO_GetPinValue       (u8 PortId,u8 PinId,u8* PinValue){       /* DEALS WITH INPUT REGISTER
                                                                     * AND IT CAN DEAL WITH OUTPUT REGISTER IF WE WANT TO MAKE SURE OF THE VALUE WE OUTPUT FOR SOME REASON */

    if( (PortId<=3) && (PinId<=7) && (PinValue!=NULL) ){

        switch(PortId){

            case DIO_PORTA:

            if (GET_BIT(PINA,PinId) == DIO_PIN_HIGH){

                *PinValue = DIO_PIN_HIGH;

            }
            else {
                *PinValue = DIO_PIN_LOW;
            }

            break;
            case DIO_PORTB:

            if (GET_BIT(PINB,PinId) == DIO_PIN_HIGH){

                *PinValue = DIO_PIN_HIGH;

            }
            else {
                *PinValue = DIO_PIN_LOW;
            }

            break;
            case DIO_PORTC:

            if (GET_BIT(PINC,PinId) == DIO_PIN_HIGH){

                *PinValue = DIO_PIN_HIGH;

            }
            else {
```

this function reads the value of pin and stores it in the variable that its address is passed to the function.



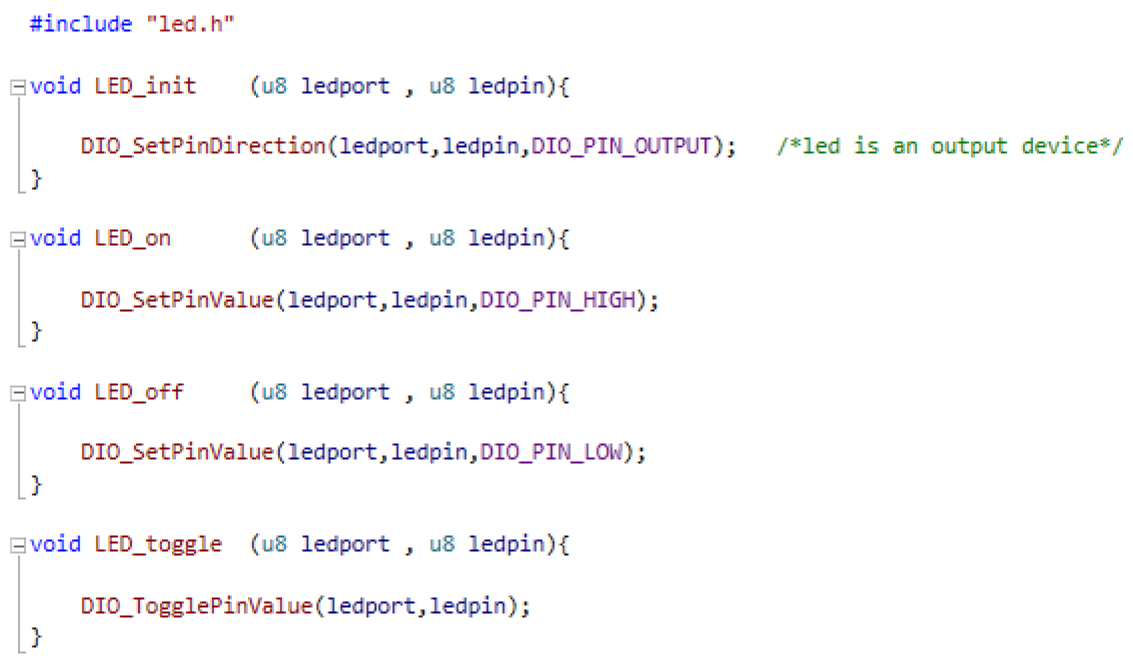
```
void DIO_TogglePinValue    (u8 PortId,u8 PinId){
    if( (PortId<=3)&&(PinId<=7) ){

        switch(PortId){

            case DIO_PORTA :

            TOG_BIT(PORTA,PinId);


            break;

            case DIO_PORTB :

            TOG_BIT(PORTB,PinId);


            break;

            case DIO_PORTC :

            TOG_BIT(PORTC,PinId);


            break;

            case DIO_PORTD :

            TOG_BIT(PORTD,PinId);


            break;


        }
    }
```

this function toggles the pin that is passed to the function



```
#ifndef LED_H_
#define LED_H_

#include "../../MCAL/DIO/DIO_interface.h"

void LED_init    (u8 ledport , u8 ledpin);
void LED_on      (u8 ledport , u8 ledpin);
void LED_off     (u8 ledport , u8 ledpin);
void LED_toggle  (u8 ledport , u8 ledpin);



#endif /* LED_H_ */
```

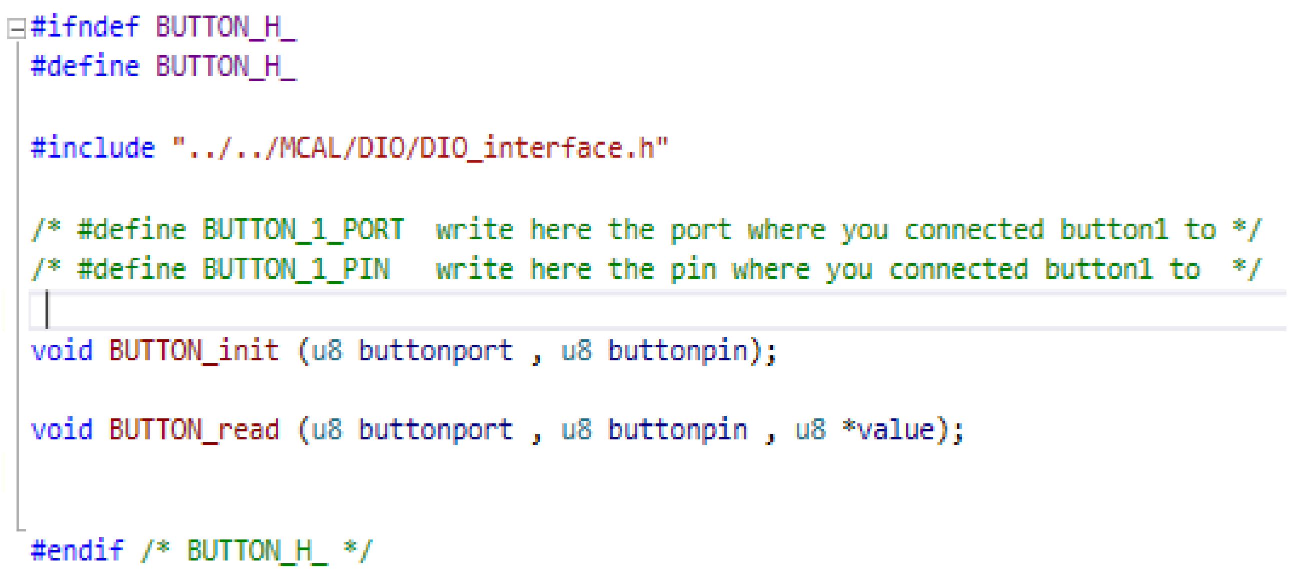
This the led.h file which contains 4 functions that are implemented in the led.c file



```
#include "led.h"

void LED_init    (u8 ledport , u8 ledpin){

    DIO_SetPinDirection(ledport,ledpin,DIO_PIN_OUTPUT);   /*led is an output device*/
}

void LED_on      (u8 ledport , u8 ledpin){

    DIO_SetPinValue(ledport,ledpin,DIO_PIN_HIGH);
}

void LED_off     (u8 ledport , u8 ledpin){

    DIO_SetPinValue(ledport,ledpin,DIO_PIN_LOW);
}

void LED_toggle  (u8 ledport , u8 ledpin){

    DIO_TogglePinValue(ledport,ledpin);
}
```

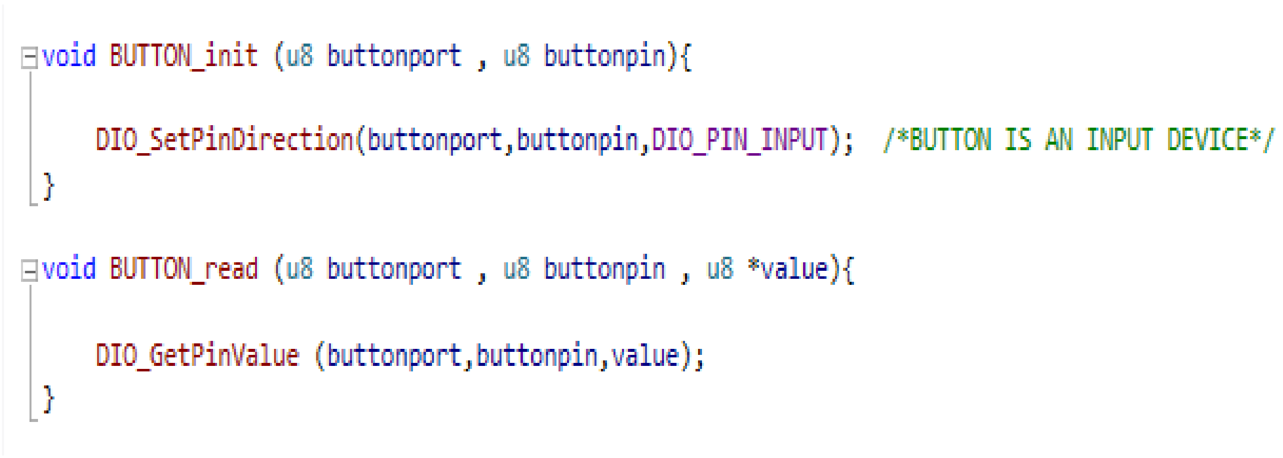
This c file implements the 4 functions which are

LED_init	sets the direction of the led to be out since it is an output device
LED_on	turns on the led
LED_off	turns off the led
LED_toggle	toggles the led



```
#ifndef BUTTON_H_
#define BUTTON_H_

#include "../../MCAL/DIO/DIO_interface.h"

/* #define BUTTON_1_PORT  write here the port where you connected button1 to */
/* #define BUTTON_1_PIN   write here the pin where you connected button1 to  */

void BUTTON_init (u8 buttonport , u8 buttonpin);

void BUTTON_read (u8 buttonport , u8 buttonpin , u8 *value);


#endif /* BUTTON_H_ */
```

this file is button.h which contains two functions prototypes that are implemented in button.c file

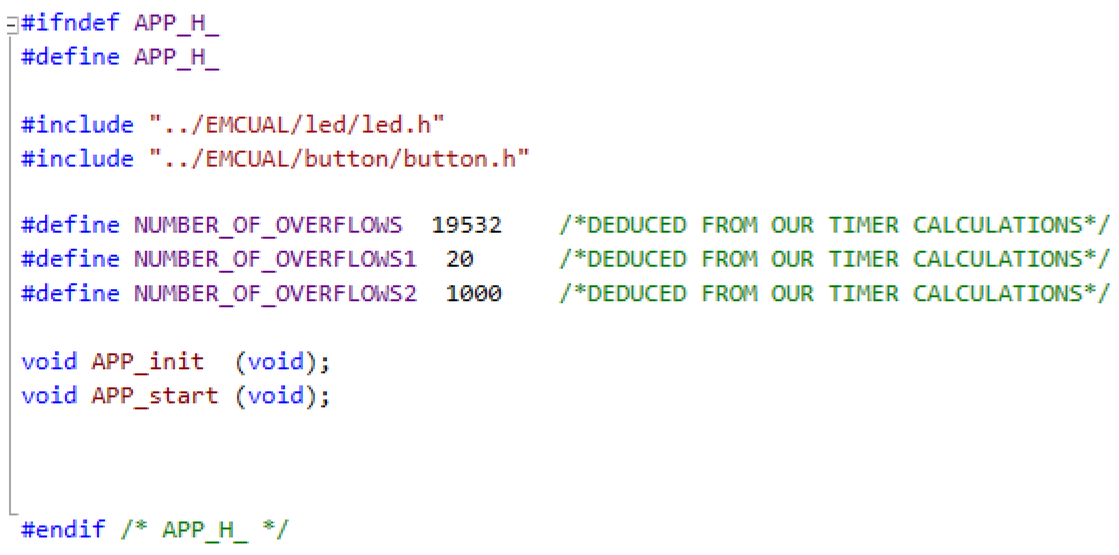


```
void BUTTON_init (u8 buttonport , u8 buttonpin){

    DIO_SetPinDirection(buttonport,buttonpin,DIO_PIN_INPUT);  /*BUTTON IS AN INPUT DEVICE*/
}

void BUTTON_read (u8 buttonport , u8 buttonpin , u8 *value){

    DIO_GetPinValue (buttonport,buttonpin,value);
}
```

this is button.c which contains the implementations of two functions

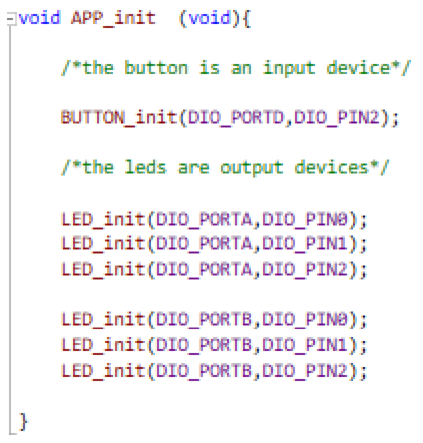
BUTTON_init   it sets the button to be an input device

BUTTON_read it reads the value of the button whether it is pressed or not



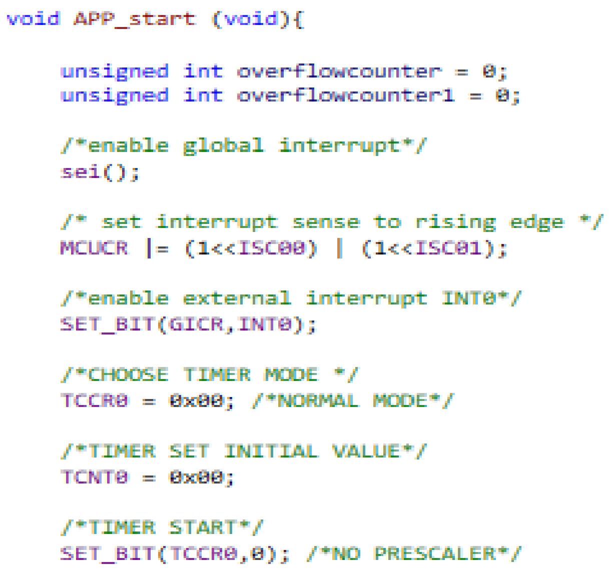
```
#ifndef APP_H_
#define APP_H_

#include "../EMCUAL/led/led.h"
#include "../EMCUAL/button/button.h"

#define NUMBER_OF_OVERFLOWS  19532    /*DEDUCED FROM OUR TIMER CALCULATIONS*/
#define NUMBER_OF_OVERFLOWS1  20      /*DEDUCED FROM OUR TIMER CALCULATIONS*/
#define NUMBER_OF_OVERFLOWS2  1000    /*DEDUCED FROM OUR TIMER CALCULATIONS*/

void APP_init  (void);
void APP_start (void);



#endif /* APP_H_ */
```

This file is app.h contains 2 functions which are APP_init and APP_start



```
void APP_init  (void){

    /*the button is an input device*/

    BUTTON_init(DIO_PORTD,DIO_PIN2);

    /*the leds are output devices*/

    LED_init(DIO_PORTA,DIO_PIN0);
    LED_init(DIO_PORTA,DIO_PIN1);
    LED_init(DIO_PORTA,DIO_PIN2);

    LED_init(DIO_PORTB,DIO_PIN0);
    LED_init(DIO_PORTB,DIO_PIN1);
    LED_init(DIO_PORTB,DIO_PIN2);

}
```

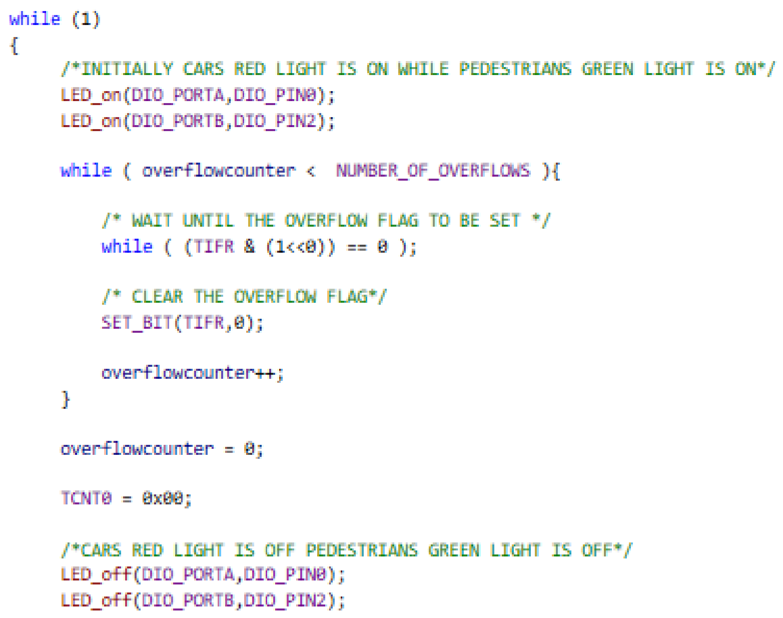
This is the implementation of the function APP_init that sets the directions of the button to be input and the 6 leds to be output



```
void APP_start (void){

    unsigned int overflowcounter = 0;
    unsigned int overflowcounter1 = 0;

    /*enable global interrupt*/
    sei();

    /* set interrupt sense to rising edge */
    MCUCR |= (1<<ISC00) | (1<<ISC01);

    /*enable external interrupt INT0*/
    SET_BIT(GICR,INT0);

    /*CHOOSE TIMER MODE */
    TCCR0 = 0x00; /*NORMAL MODE*/

    /*TIMER SET INITIAL VALUE*/
    TCNT0 = 0x00;

    /*TIMER START*/
    SET_BIT(TCCR0,0); /*NO PRESCALER*/
```

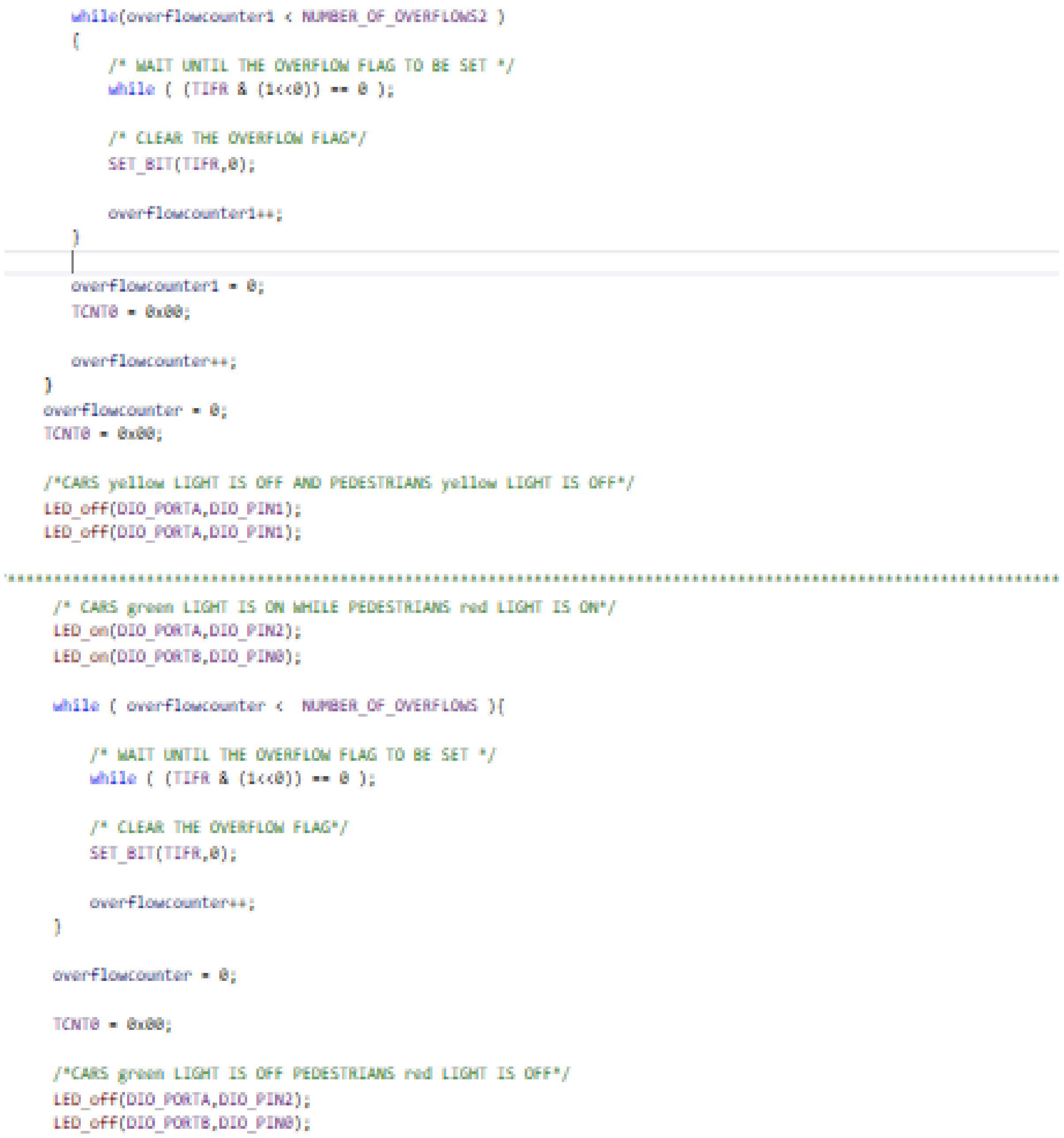
this is the first part of the implementation of APP_start function

this part handles the interrupt and the timer



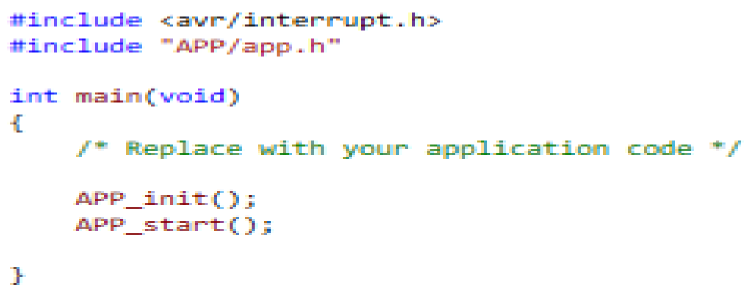
```
while (1)
{
    /*INITIALLY CARS RED LIGHT IS ON WHILE PEDESTRIANS GREEN LIGHT IS ON*/
    LED_on(DIO_PORTA,DIO_PIN0);
    LED_on(DIO_PORTB,DIO_PIN2);

    while ( overflowcounter <  NUMBER_OF_OVERFLOWS ){

        /* WAIT UNTIL THE OVERFLOW FLAG TO BE SET */
        while ( (TIFR & (1<<0)) == 0 );

        /* CLEAR THE OVERFLOW FLAG*/
        SET_BIT(TIFR,0);

        overflowcounter++;
    }

    overflowcounter = 0;

    TCNT0 = 0x00;

    /*CARS RED LIGHT IS OFF PEDESTRIANS GREEN LIGHT IS OFF*/
    LED_off(DIO_PORTA,DIO_PIN0);
    LED_off(DIO_PORTB,DIO_PIN2);
```

This part lights the cars’ red light on and pedestrians’ green light on



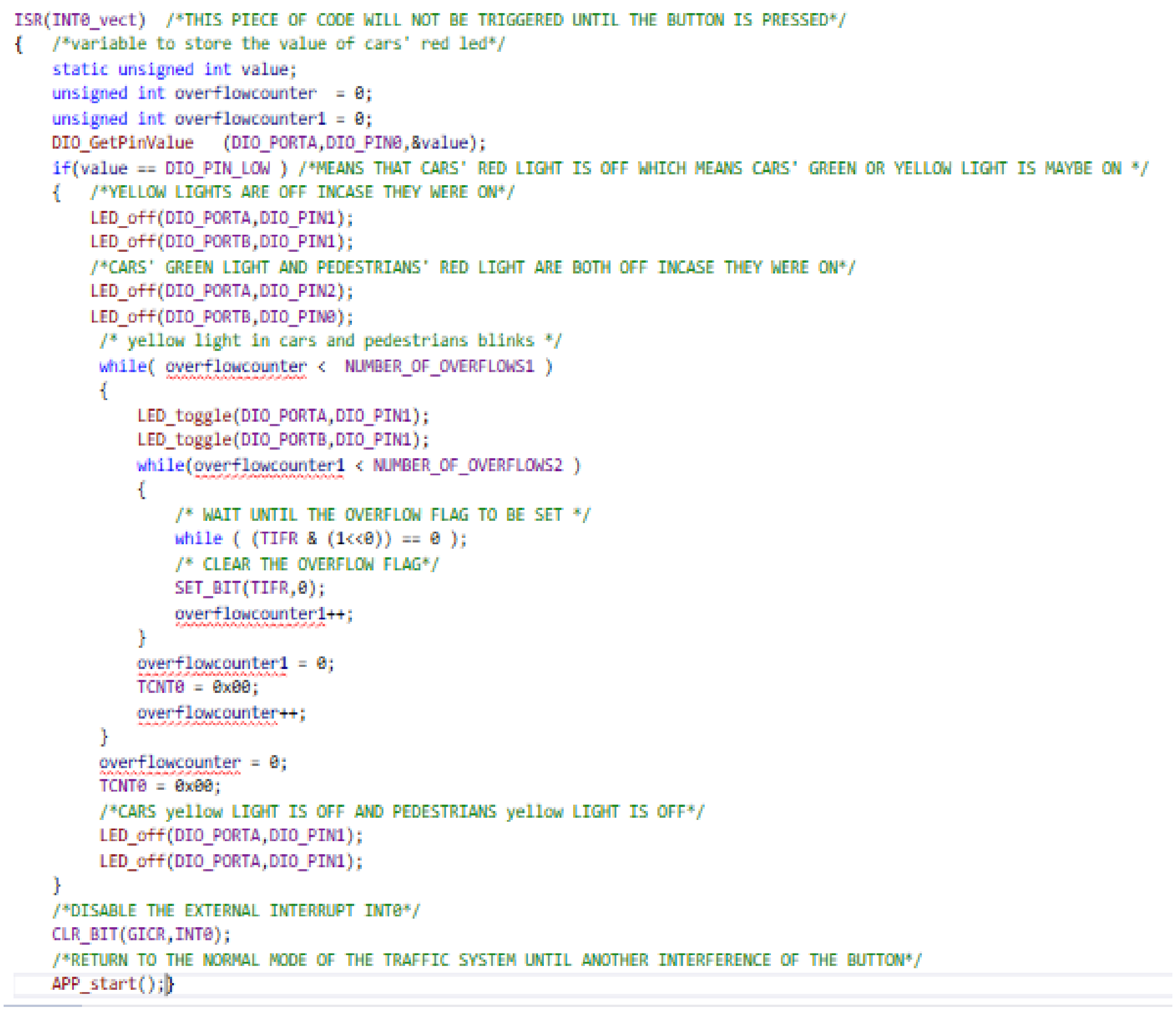
```
        while(overflowcounter1 < NUMBER_OF_OVERFLOWS2 )
        {
            /* WAIT UNTIL THE OVERFLOW FLAG TO BE SET */
            while ( (TIFR & (1<<0)) == 0 );

            /* CLEAR THE OVERFLOW FLAG*/
            SET_BIT(TIFR,0);

            overflowcounter1++;
        }

        overflowcounter1 = 0;
        TCNT0 = 0x00;

        overflowcounter++;
    }
    overflowcounter = 0;
    TCNT0 = 0x00;

    /*CARS yellow LIGHT IS OFF AND PEDESTRIANS yellow LIGHT IS OFF*/
    LED_off(DIO_PORTA,DIO_PIN1);
    LED_off(DIO_PORTA,DIO_PIN1);

**********************************************************************************************
     /* CARS green LIGHT IS ON WHILE PEDESTRIANS red LIGHT IS ON*/
     LED_on(DIO_PORTA,DIO_PIN2);
     LED_on(DIO_PORTB,DIO_PIN0);

     while ( overflowcounter <  NUMBER_OF_OVERFLOWS ){

         /* WAIT UNTIL THE OVERFLOW FLAG TO BE SET */
         while ( (TIFR & (1<<0)) == 0 );

         /* CLEAR THE OVERFLOW FLAG*/
         SET_BIT(TIFR,0);

         overflowcounter++;
     }

     overflowcounter = 0;

     TCNT0 = 0x00;

     /*CARS green LIGHT IS OFF PEDESTRIANS red LIGHT IS OFF*/
     LED_off(DIO_PORTA,DIO_PIN2);
     LED_off(DIO_PORTB,DIO_PIN0);
```

This part makes the yellow light of both blinks then after 5 seconds the cars'green light is on while the pedestrians' red light is on



```
#include <avr/interrupt.h>
#include "APP/app.h"

int main(void)
{
    /* Replace with your application code */

    APP_init();
    APP_start();

}
```

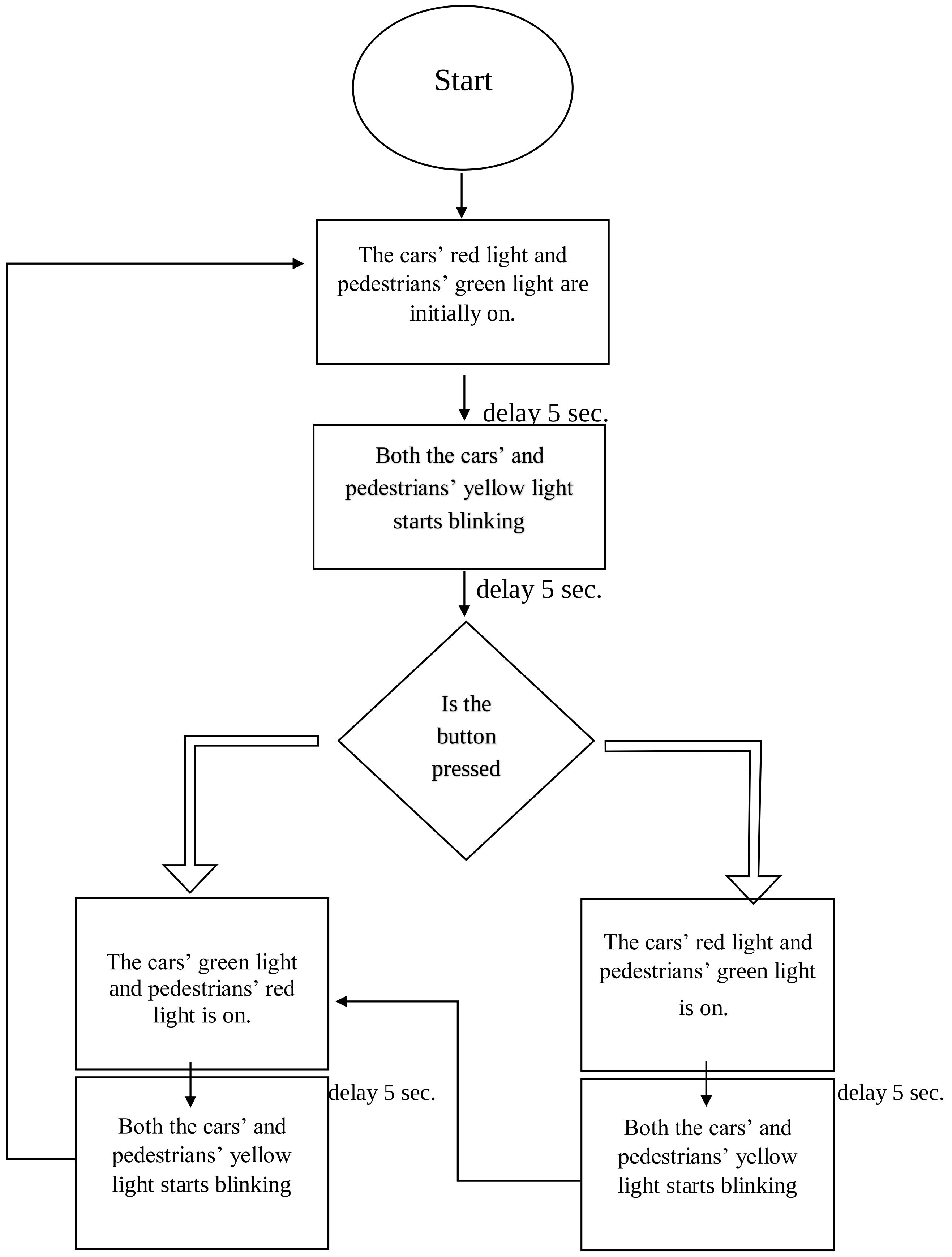
this is part of the main.c where we call the two functions in the app layer



```
ISR(INT0_vect)  /*THIS PIECE OF CODE WILL NOT BE TRIGGERED UNTIL THE BUTTON IS PRESSED*/
{   /*variable to store the value of cars' red led*/
    static unsigned int value;
    unsigned int overflowcounter  = 0;
    unsigned int overflowcounter1 = 0;
    DIO_GetPinValue   (DIO_PORTA,DIO_PIN0,&value);
    if(value == DIO_PIN_LOW ) /*MEANS THAT CARS' RED LIGHT IS OFF WHICH MEANS CARS' GREEN OR YELLOW LIGHT IS MAYBE ON */
    {   /*YELLOW LIGHTS ARE OFF INCASE THEY WERE ON*/
        LED_off(DIO_PORTA,DIO_PIN1);
        LED_off(DIO_PORTB,DIO_PIN1);
        /*CARS' GREEN LIGHT AND PEDESTRIANS' RED LIGHT ARE BOTH OFF INCASE THEY WERE ON*/
        LED_off(DIO_PORTA,DIO_PIN2);
        LED_off(DIO_PORTB,DIO_PIN0);
         /* yellow light in cars and pedestrians blinks */
         while( overflowcounter <  NUMBER_OF_OVERFLOWS1 )
         {
             LED_toggle(DIO_PORTA,DIO_PIN1);
             LED_toggle(DIO_PORTB,DIO_PIN1);
             while(overflowcounter1 < NUMBER_OF_OVERFLOWS2 )
             {
                 /* WAIT UNTIL THE OVERFLOW FLAG TO BE SET */
                 while ( (TIFR & (1<<0)) == 0 );
                 /* CLEAR THE OVERFLOW FLAG*/
                 SET_BIT(TIFR,0);
                 overflowcounter1++;
             }
             overflowcounter1 = 0;
             TCNT0 = 0x00;
             overflowcounter++;
         }
         overflowcounter = 0;
         TCNT0 = 0x00;
         /*CARS yellow LIGHT IS OFF AND PEDESTRIANS yellow LIGHT IS OFF*/
         LED_off(DIO_PORTA,DIO_PIN1);
         LED_off(DIO_PORTA,DIO_PIN1);
    }
    /*DISABLE THE EXTERNAL INTERRUPT INT0*/
    CLR_BIT(GICR,INT0);
    /*RETURN TO THE NORMAL MODE OF THE TRAFFIC SYSTEM UNTIL ANOTHER INTERFERENCE OF THE BUTTON*/
    APP_start();}
```

this is the ISR which is related to the interrupts and is only triggered when the button is pressed and The function checks first for the state of pin 0 in port A (cars' red light). If the red light is on nothing will happen. If the green or yellow light is on then the yellow light starts blinking for 5 seconds then the cars' red light and pedestrians green light is turned on.

At the end the external interrupt is disabled to return to normal mode and the function app start is called in order for the traffic light system to work normally.


Start
The cars' red light and pedestrians' green light are initially on.
delay 5 sec.
Both the cars' and pedestrians' yellow light starts blinking
delay 5 sec.
Is the button pressed
The cars' green light and pedestrians' red light is on.
delay 5 sec.
Both the cars' and pedestrians' yellow light starts blinking
The cars' red light and pedestrians' green light is on.
delay 5 sec.
Both the cars' and pedestrians' yellow light starts blinking