On-demand Traffic Light Control System

SEPTEMBER 15

egFWD Embedded Systems professional track

Authored by: Adel Mostafa Kamel Helal

1. System description

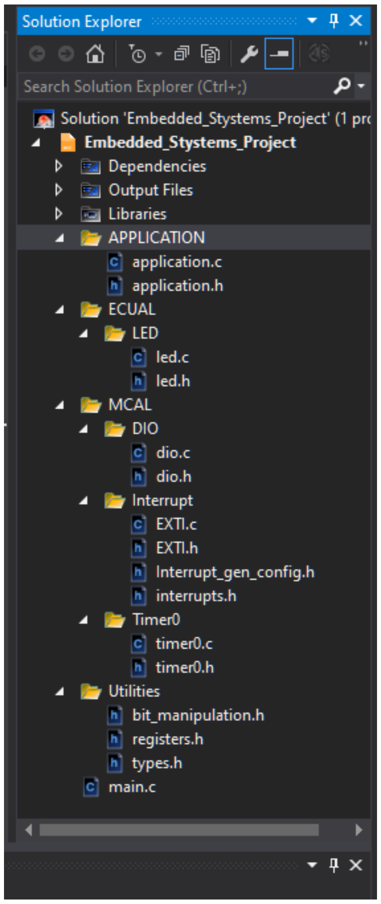
Traffic lights are signaling devices positioned at road intersections and pedestrian crossings to control the flow of traffic.
Traffic lights consists of three signals transmitting meaningful information through colors to the cars' drivers and the pedestrians.
The traffic light colors are red, yellow, and green arranged vertically.
Crosswalk button lets the signal controller know that someone wants to cross the street, so the light signals adjusts, giving the pedestrian enough time to get across.

2. System Design

Software Design

Project's folder structure is layered architecture with four layers:

1. **Application layer:**
   Has the application code which resides in top.
2. **Electrical Unit Abstraction Layer(ECUAL) layer:**
   Contains the **LED** driver.
3. **Micro Controller Abstraction Layer(MCAL) layer:**
   Contains the **DIO**, **timer0** and **interrupt** drivers.
4. **Utilities layer:**
   Contains helper header files to be support the other drivers. And it has **types.h**, **registers.h** and **bit_manipulation.h** header files.



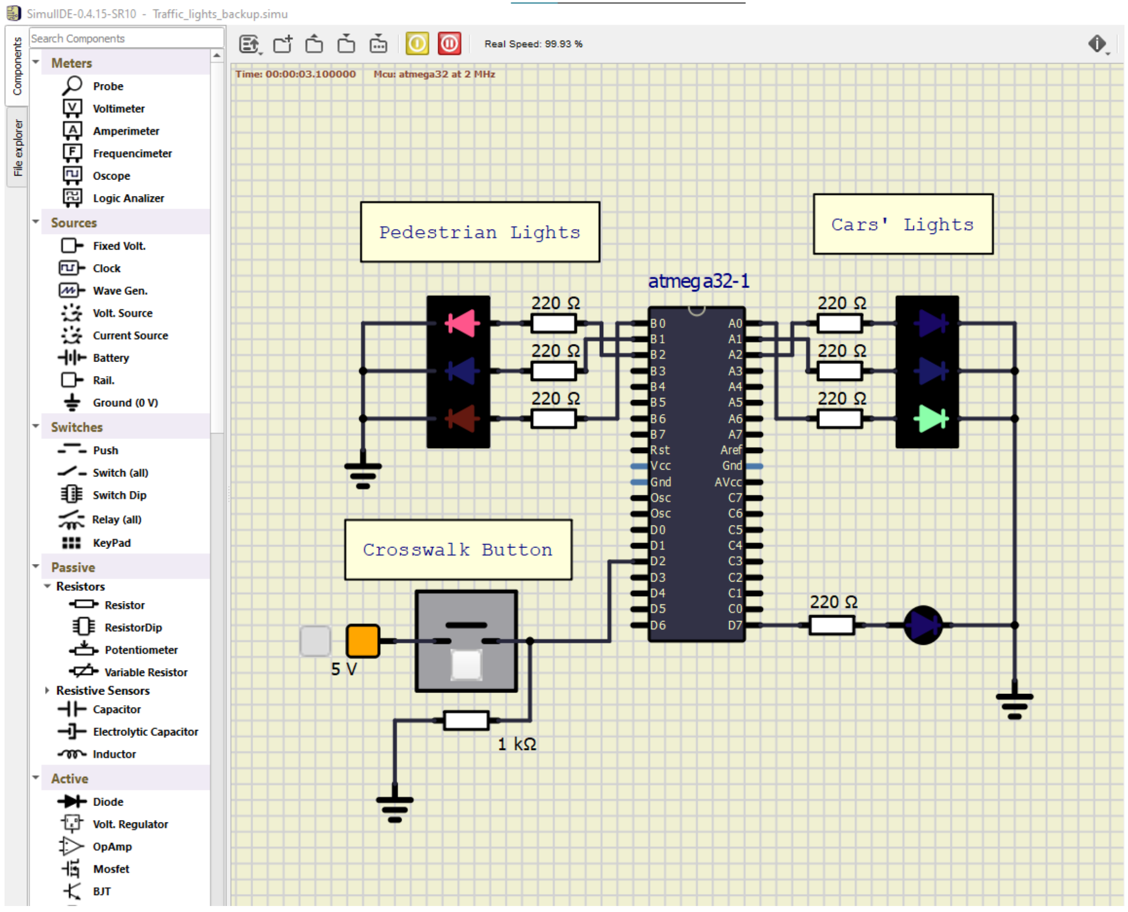
```
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Embedded_Stystems_Project' (1 pro
  Embedded_Stystems_Project
    Dependencies
    Output Files
    Libraries
    APPLICATION
      application.c
      application.h
    ECUAL
      LED
        led.c
        led.h
    MCAL
      DIO
        dio.c
        dio.h
      Interrupt
        EXTI.c
        EXTI.h
        Interrupt_gen_config.h
        interrupts.h
      Timer0
        timer0.c
        timer0.h
    Utilities
      bit_manipulation.h
      registers.h
      types.h
    main.c
```

Each driver contains the used **APIs** and **data types** declared in the driver header file, defined and used in the source file.

Hardware Design

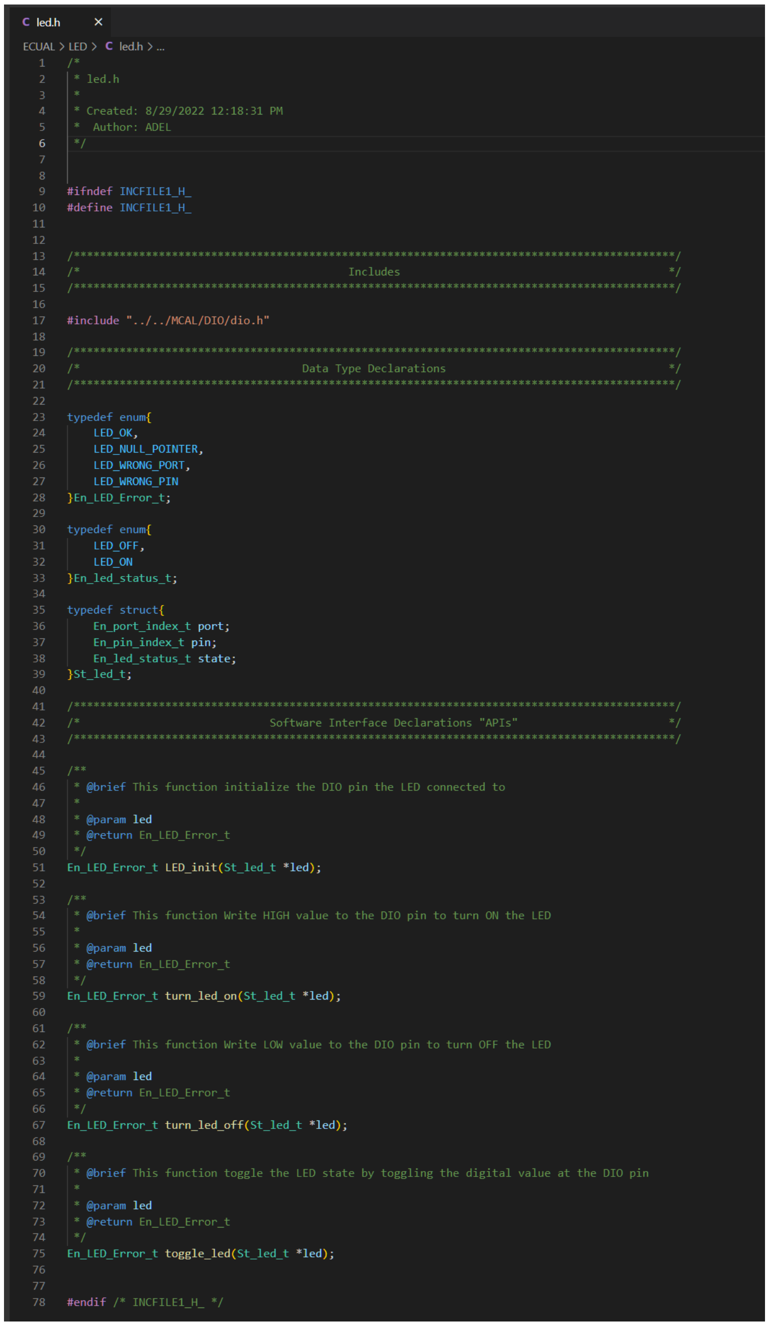
- Atmega32 microcontroller is used with three LEDs for cars signals connected on **Port A** (Green to **Pin 0**, Yellow to **Pin 1** and Red to **Pin 2**), three other LEDs for pedestrian signals connected on **Port B** (Green to **Pin 0**, Yellow to **Pin 1** and Red to **Pin 2**) and one push button connected to **INT0** Pin (**Port D**, **Pin 2**) for pedestrian. And another one connected to (**Port D, Pin 7**) for indicating that an error occurred when testing a driver API manually.

- Simulation is done using SimulIDE software to simulate the traffic light control.



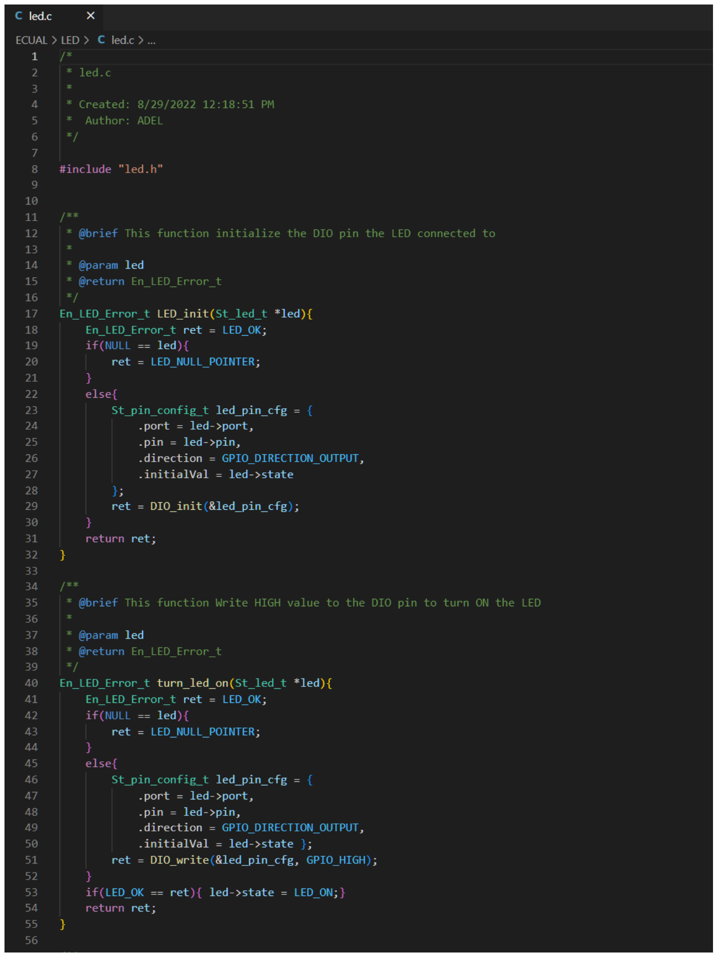
Implementation

First implemented the MCAL drivers i.e. DIO, Timer0 and Interrupt drivers the I Implemented the ECUAL LED driver. I wrote a skeleton for each function using comments and then converted them into the appropriate code.
Each driver has at least two files, source file and header file.
The **.h header file** consists of the inclusion section of the needed header files of the lower layer, macros, and functions like macro, new defined data types and the driver APIs prototypes.



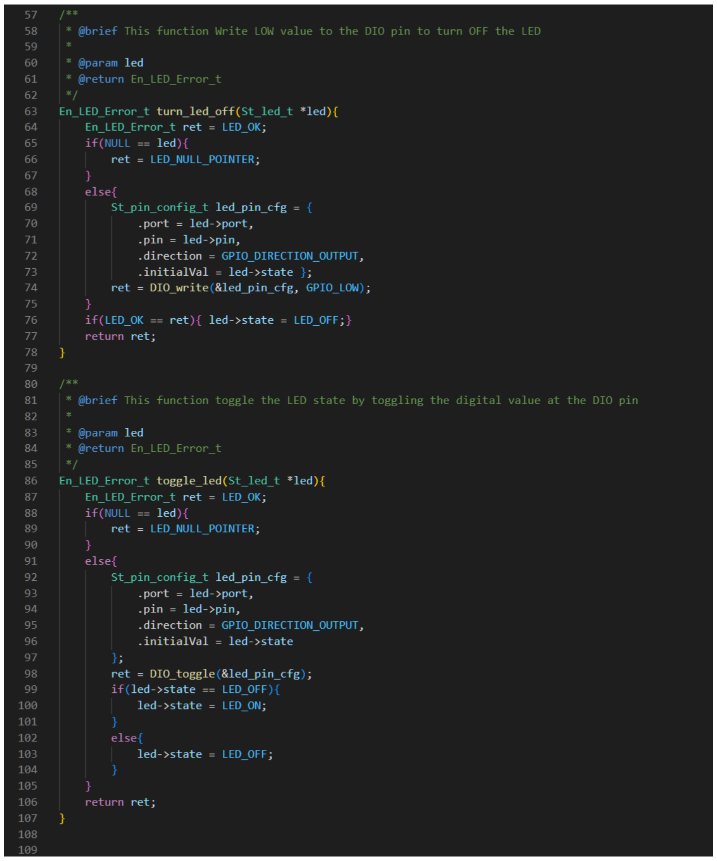
```c
/*
 * led.h
 *
 * Created: 8/29/2022 12:18:31 PM
 *  Author: ADEL
 */


#ifndef INCFILE1_H_
#define INCFILE1_H_


/*******************************************************************************************/
/*                                           Includes                                      */
/*******************************************************************************************/

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

/*******************************************************************************************/
/*                                   Data Type Declarations                                */
/*******************************************************************************************/

typedef enum{
    LED_OK,
    LED_NULL_POINTER,
    LED_WRONG_PORT,
    LED_WRONG_PIN
}En_LED_Error_t;

typedef enum{
    LED_OFF,
    LED_ON
}En_led_status_t;

typedef struct{
    En_port_index_t port;
    En_pin_index_t pin;
    En_led_status_t state;
}St_led_t;

/*******************************************************************************************/
/*                              Software Interface Declarations "APIs"                     */
/*******************************************************************************************/

/**
 * @brief This function initialize the DIO pin the LED connected to
 *
 * @param led
 * @return En_LED_Error_t
 */
En_LED_Error_t LED_init(St_led_t *led);

/**
 * @brief This function Write HIGH value to the DIO pin to turn ON the LED
 *
 * @param led
 * @return En_LED_Error_t
 */
En_LED_Error_t turn_led_on(St_led_t *led);

/**
 * @brief This function Write LOW value to the DIO pin to turn OFF the LED
 *
 * @param led
 * @return En_LED_Error_t
 */
En_LED_Error_t turn_led_off(St_led_t *led);

/**
 * @brief This function toggle the LED state by toggling the digital value at the DIO pin
 *
 * @param led
 * @return En_LED_Error_t
 */
En_LED_Error_t toggle_led(St_led_t *led);


#endif /* INCFILE1_H_ */
```

The driver **.c source file** includes the corresponding driver header file the definition of the driver functions.


led.c ×

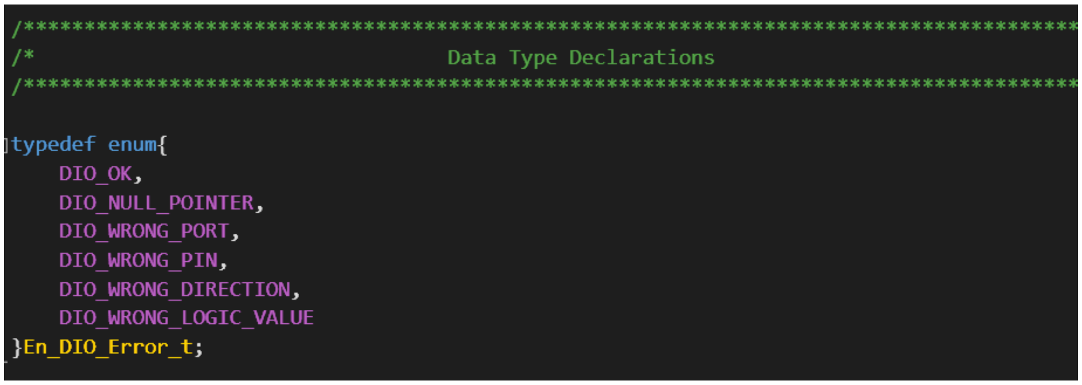
ECUAL > LED > led.c > ...

```c
/*
 * led.c
 *
 * Created: 8/29/2022 12:18:51 PM
 *  Author: ADEL
 */

#include "led.h"


/**
 * @brief This function initialize the DIO pin the LED connected to
 *
 * @param led
 * @return En_LED_Error_t
 */
En_LED_Error_t LED_init(St_led_t *led){
    En_LED_Error_t ret = LED_OK;
    if(NULL == led){
        ret = LED_NULL_POINTER;
    }
    else{
        St_pin_config_t led_pin_cfg = {
            .port = led->port,
            .pin = led->pin,
            .direction = GPIO_DIRECTION_OUTPUT,
            .initialVal = led->state
        };
        ret = DIO_init(&led_pin_cfg);
    }
    return ret;
}

/**
 * @brief This function Write HIGH value to the DIO pin to turn ON the LED
 *
 * @param led
 * @return En_LED_Error_t
 */
En_LED_Error_t turn_led_on(St_led_t *led){
    En_LED_Error_t ret = LED_OK;
    if(NULL == led){
        ret = LED_NULL_POINTER;
    }
    else{
        St_pin_config_t led_pin_cfg = {
            .port = led->port,
            .pin = led->pin,
            .direction = GPIO_DIRECTION_OUTPUT,
            .initialVal = led->state };
        ret = DIO_write(&led_pin_cfg, GPIO_HIGH);
    }
    if(LED_OK == ret){ led->state = LED_ON;}
    return ret;
}

```



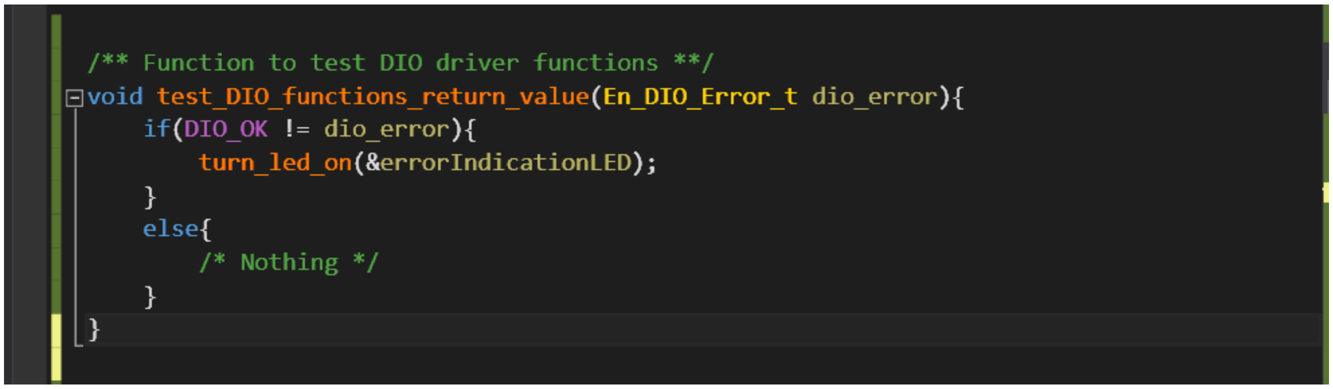
```c
/**
 * @brief This function Write LOW value to the DIO pin to turn OFF the LED
 *
 * @param led
 * @return En_LED_Error_t
 */
En_LED_Error_t turn_led_off(St_led_t *led){
    En_LED_Error_t ret = LED_OK;
    if(NULL == led){
        ret = LED_NULL_POINTER;
    }
    else{
        St_pin_config_t led_pin_cfg = {
            .port = led->port,
            .pin = led->pin,
            .direction = GPIO_DIRECTION_OUTPUT,
            .initialVal = led->state };
        ret = DIO_write(&led_pin_cfg, GPIO_LOW);
    }
    if(LED_OK == ret){ led->state = LED_OFF;}
    return ret;
}

/**
 * @brief This function toggle the LED state by toggling the digital value at the DIO pin
 *
 * @param led
 * @return En_LED_Error_t
 */
En_LED_Error_t toggle_led(St_led_t *led){
    En_LED_Error_t ret = LED_OK;
    if(NULL == led){
        ret = LED_NULL_POINTER;
    }
    else{
        St_pin_config_t led_pin_cfg = {
            .port = led->port,
            .pin = led->pin,
            .direction = GPIO_DIRECTION_OUTPUT,
            .initialVal = led->state
        };
        ret = DIO_toggle(&led_pin_cfg);
        if(led->state == LED_OFF){
            led->state = LED_ON;
        }
        else{
            led->state = LED_OFF;
        }
    }
    return ret;
}

```

Enum data type is implemented for each driver and each function returns error state to indicate the status of the function process is it done correctly or there was an error occurred.



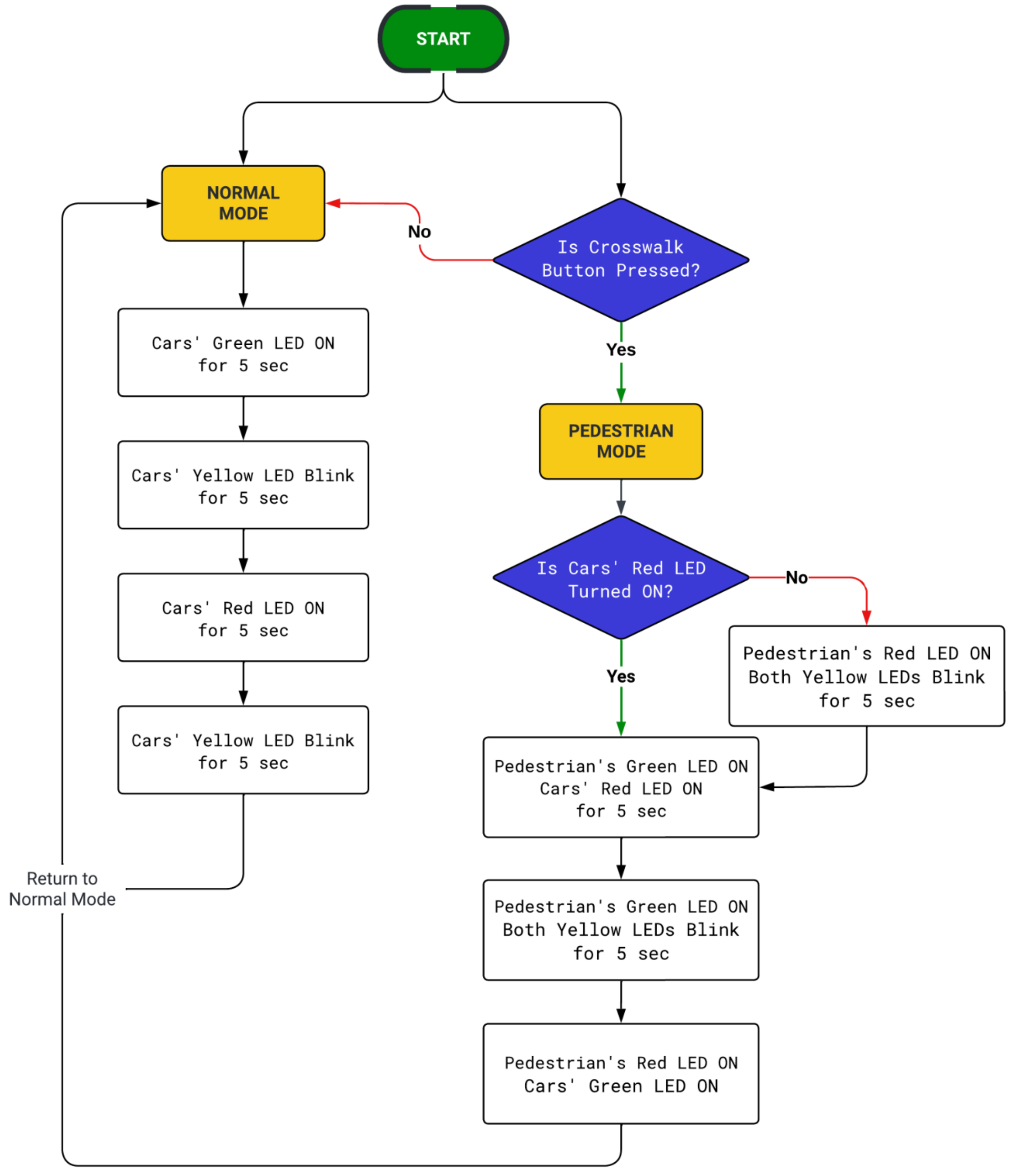
```
/*****************************************************************************
/*                                   Data Type Declarations
/*****************************************************************************

typedef enum{
    DIO_OK,
    DIO_NULL_POINTER,
    DIO_WRONG_PORT,
    DIO_WRONG_PIN,
    DIO_WRONG_DIRECTION,
    DIO_WRONG_LOGIC_VALUE
}En_DIO_Error_t;
```

In the main function you can test specific driver's functions manually through the driver test module implemented in the main source file by passing the return value of the driver function to the test module and if this value isn't **OK** the function will turn the error indication LED.



```
/** Function to test DIO driver functions **/
void test_DIO_functions_return_value(En_DIO_Error_t dio_error){
    if(DIO_OK != dio_error){
        turn_led_on(&errorIndicationLED);
    }
    else{
        /* Nothing */
    }
}
```

3. System Flow Chart


START
NORMAL MODE
Cars' Green LED ON for 5 sec
Cars' Yellow LED Blink for 5 sec
Cars' Red LED ON for 5 sec
Cars' Yellow LED Blink for 5 sec
Return to Normal Mode
No
Is Crosswalk Button Pressed?
Yes
PEDESTRIAN MODE
Is Cars' Red LED Turned ON?
No
Pedestrian's Red LED ON Both Yellow LEDs Blink for 5 sec
Yes
Pedestrian's Green LED ON Cars' Red LED ON for 5 sec
Pedestrian's Green LED ON Both Yellow LEDs Blink for 5 sec
Pedestrian's Red LED ON Cars' Green LED ON

4. System Constraints

There is no constrains on the system.

5. Project Video

1- **Project overview**

https://github.com/adelmostafa389/On_Demand_Traffic_Light_Control_System/blob/main/02-Recordings/01-Overview_video.mp4

2- **Project structure**

https://github.com/adelmostafa389/On_Demand_Traffic_Light_Control_System/blob/main/02-Recordings/02-Project_structure.mp4

3- **Project implementation**

https://github.com/adelmostafa389/On_Demand_Traffic_Light_Control_System/blob/main/02-Recordings/03-Project_impelemntation.mp4

4- **Driver APIs test**

https://github.com/adelmostafa389/On_Demand_Traffic_Light_Control_System/blob/main/02-Recordings/04-led-driver-test.mp4

5- **Application test simulation**

https://github.com/adelmostafa389/On_Demand_Traffic_Light_Control_System/blob/main/02-Recordings/5_APPLICATION_Test_Simulation.mp4