On-Demand Traffic Light Control Project

1- System Description

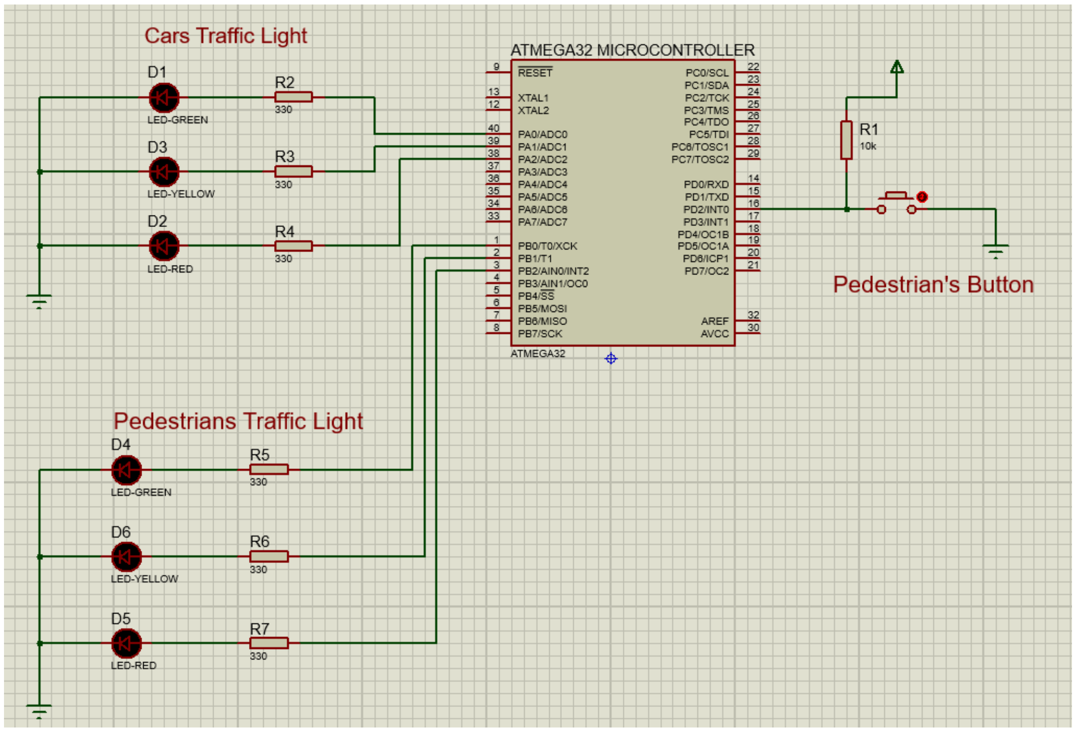
This system is called On-Demand Traffic Light Control, we meet this traffic light every day in the streets through this documentation we will discuss what it is and how we made it.

This project is based on AVR microcontroller especially Atmega32 and we considered it the brain of our project that takes all the decisions, we also used some hardware components such as LEDs and push button to simulate the traffic light as in real world.

The hardware requirements:

1. ATmega32 microcontroller
2. One push button connected to INT0 pin for pedestrian
3. Three LEDs for cars - Green, Yellow, and Red, connected on port A, pins 0, 1, and 2
4. Three LEDs for pedestrians - Green, Yellow, and Red, connected on port B, pins 0, 1, and 2

As shown from the simulation using proteus



Software requirements:
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.

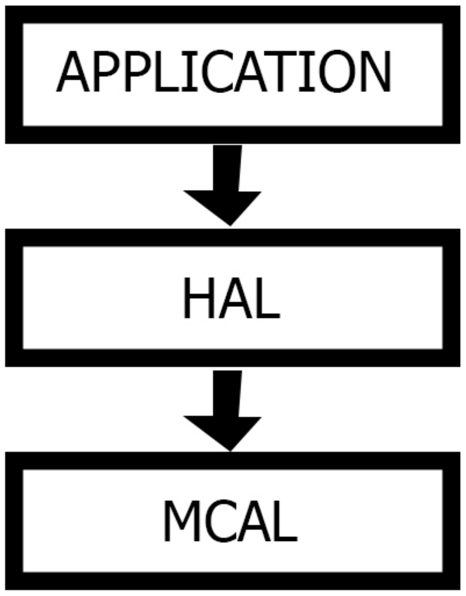
All the above is the system description and what it is consists of.

2- System Design

In the section we will see how I design my system following the SOLID principles.

We have 3 layers which they are as follows from down to up:

1. MCAL which consists of GPIO and Timer drivers
2. ECUAL which consists of Button and Led drivers
3. Application layer which contains the flow of the system through its states



Let’s discuss every driver on brief,

GPIO Driver:

Simply, it contains some functions that helps us in setting pins direction, writing on pins, and reading their values and so on.

GPIO Driver API's:

- void GPIO_setupPinDirection (uint8 port_num, uint8 pin_num, GPIO_PinDirectionType direction);

Description:

Setup the direction of the required pin input/output. If the input port number or pin number are not correct, the function will not handle the request.

- void GPIO_writePin (uint8 port_num, uint8 pin_num, uint8 value);

Description:

Write the value Logic High or Logic Low on the required pin. If the input port number or pin number are not correct, the function will not handle the request. If the pin is input, this function will enable/disable the internal pull-up resistor.

- uint8 GPIO_readPin (uint8 port_num, uint8 pin_num);

Description:

Read and return 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 Logic Low.

- void GPIO_setupPortDirection (uint8 port_num, uint8 direction);

Description:

Setup the direction of the required port all pins input/output. If the direction value is PORT_INPUT all pins in this port should be input pins. If the direction value is PORT_OUTPUT all pins in this port should be output pins. If the input port number is not correct, the function will not handle the request.

- void GPIO_writePort (uint8 port_num, uint8 value);

Description:

Write the value on the required port. If any pin in the port is output pin the value will be written. If any pin in the port is input pin this will activate/deactivate the internal pull-up resistor. If the input port number is not correct, the function will not handle the request.

- uint8 GPIO_readPort (uint8 port_num);

Description:

Read and return the value of the required port. If the input port number is not correct, the function will return ZERO value.

Timer Driver:

It contains some functions to control the timer driver through them such as timer init and timer stop.

Timer Driver API's:

- void Timer0_init(const TIMER_ConfigType * Config_Ptr);

Description:

Function to Initialize Timer Driver, Work in Interrupt Mode, Timer initial value, Timer_Mode (OverFlow, Compare), if using CTC mode: Timer compare match, and Timer_Prescaler.

- void timer_setCallBack(void(*a_ptr)(void));

Description: Function to set the Call Back function address.

- void Timer_stop();

Description: Function to stop the clock of the timer to stop incrementing.

- void Timer_DeInit();

Description: Function to DeInit the timer to start again from beginning.

LED Driver API's:

- void LED_init(uint8 ledPort, uint8 ledPin);
- void LED_on(uint8 ledPort, uint8 ledPin);
- void LED_off(uint8 ledPort, uint8 ledPin);
- void LED_toggle(uint8 ledPort, uint8 ledPin);
- uint8 LED_read(uint8 ledPort, uint8 ledPin);

Button Driver API's:

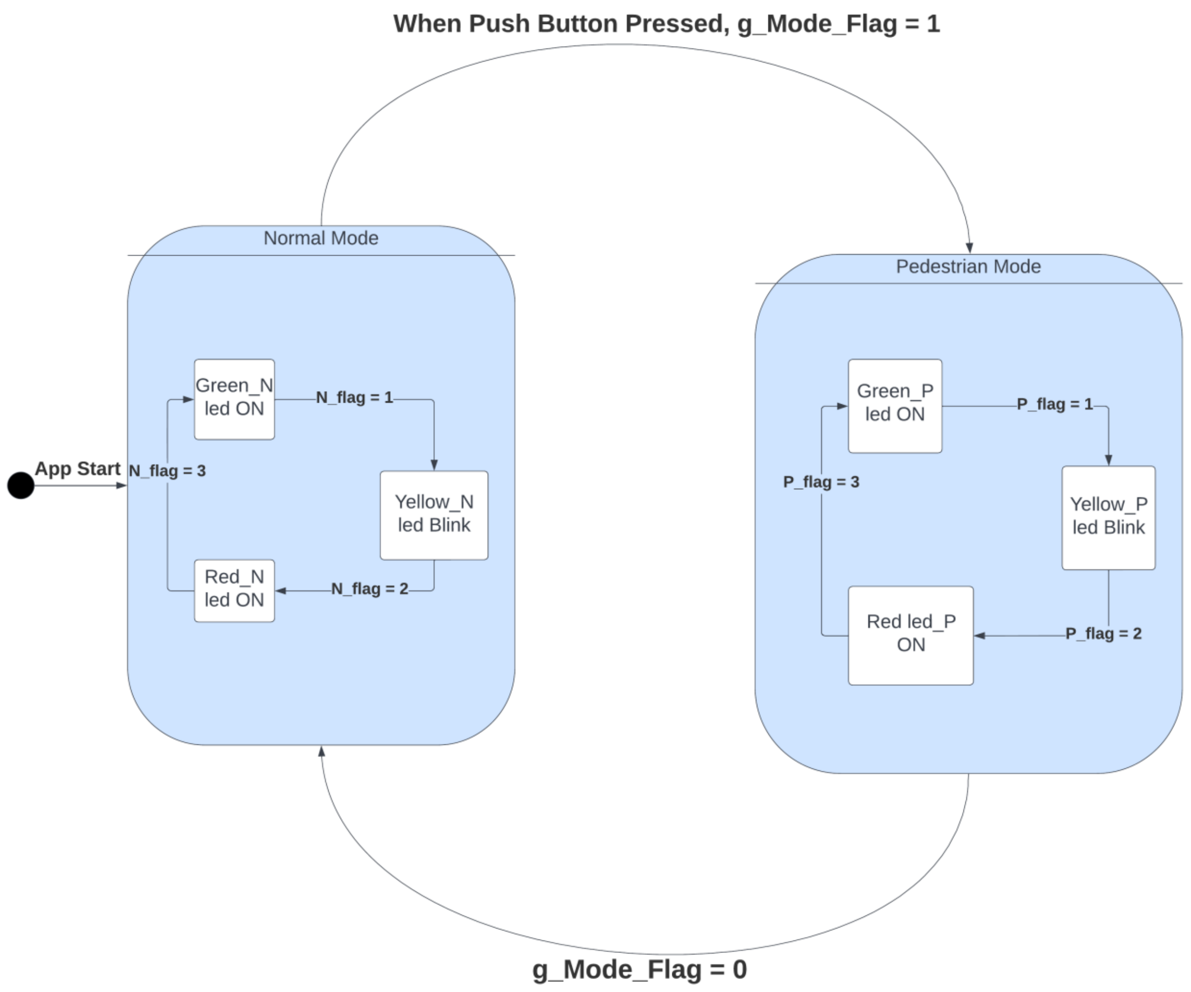
- void BUTTON_init(uint8 buttonPort, uint8 buttonPin);
- void BUTTON_read(uint8 buttonPort, uint8 buttonPin, uint8 *value);

Application API's:

- void TL_5SEC_Tick_Processing(void);
- void TL_SEC_Tick_Processing(void);
- void TL_SEC(void);
- void normal_mode(void);
- void TL_5SEC(void);
- void pedestrian_mode(void);

After finishing all the drivers, I started writing the main flow of the system as that will be discussed in the next section.

3- System Flowchart



4- System Constrains

N/A, the system is operating as requested in the requirements.