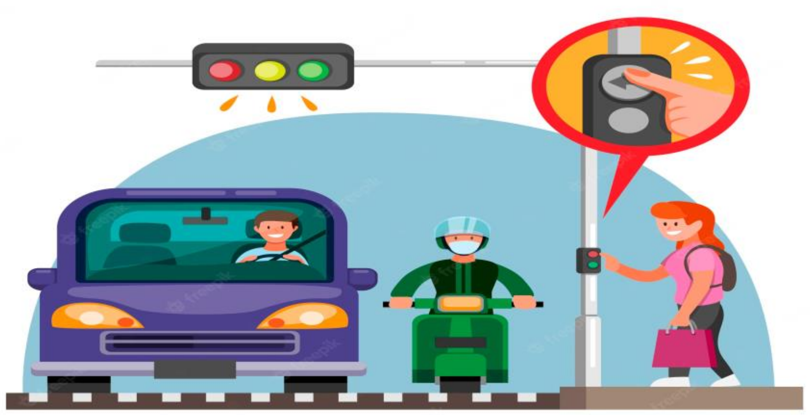
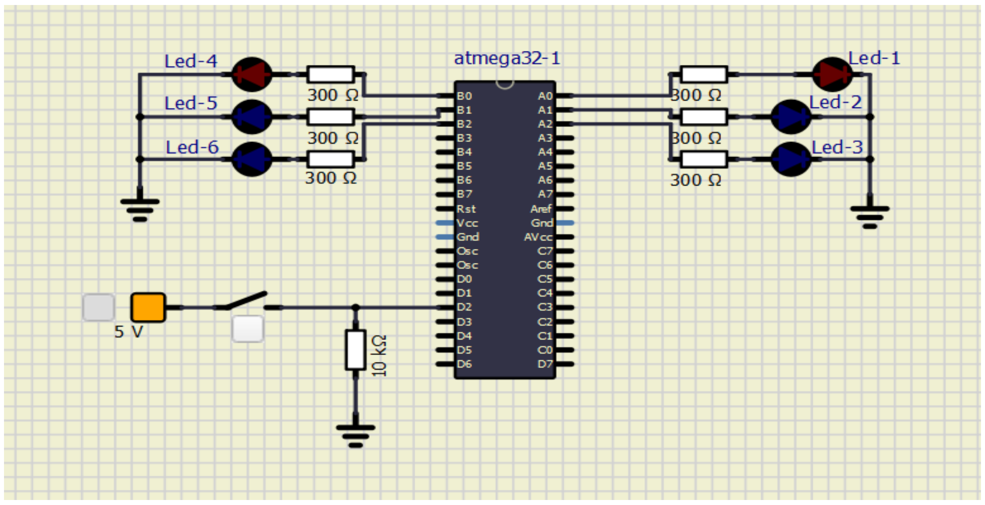
System Description

The system consists of a total of 6 led lights, 3 for car traffic lights and 3 for pedestrian traffic lights. A push button is used by pedestrians to request to cross the street. Normal flow of the system is that the car traffic lights will change every five seconds starting from green to blinking yellow to red then back to yellow again while the pedestrian traffic lights operate opposite to the cars one. This flow will keep working until someone press on the pedestrian push button and pedestrian mode will start operating. Pedestrian mode operates as follows:

- 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.
- 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.
- 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.
- 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.
- Traffic lights signals are going to the normal mode again.

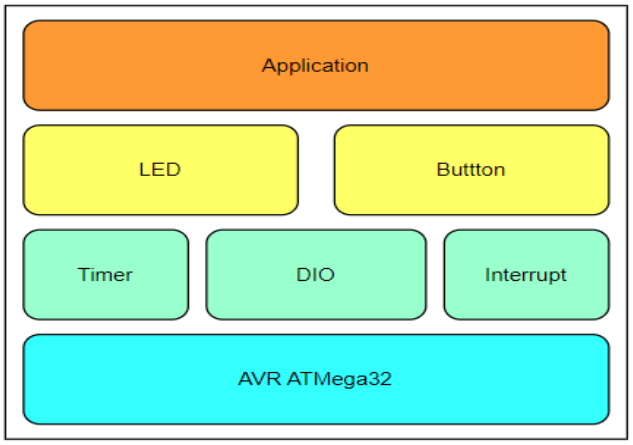


AVR ATmega32 microcontroller is used to control this system and the schematic is as the below figure. For this project, Microchip Studio is used for development while SimulIDE is used for simulations.



System Design

System is designed based on the following layered architecture. SOLID principles and layers abstraction was respected to allow the system elements to be reusable, upgradable and maintainable.



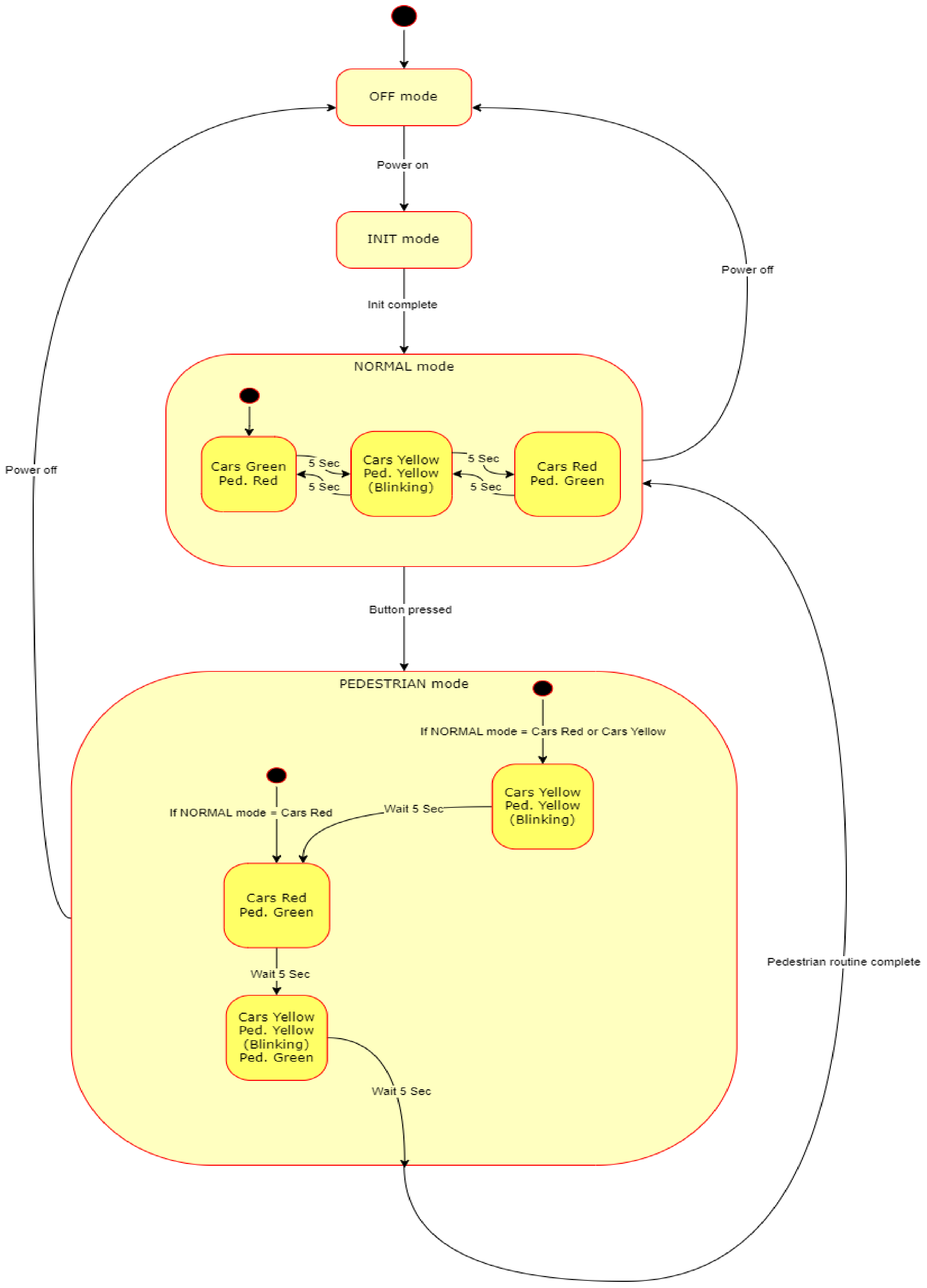
The system is divided into 4 abstracted layers with their corresponding APIs:

1- Application Layer:
   a. `void App_start(void)`
   b. `void App_init(void)`

2- ECUAL (ECU Abstraction layer)
   a. LED
      i. `void LED_init(uint8_t ledPort, uint8_t ledPin)`
      ii. `void LED_on(uint8_t ledPort, uint8_t ledPin)`
      iii. `void LED_off(uint8_t ledPort, uint8_t ledPin)`
      iv. `void LED_toggle(uint8_t ledPort, uint8_t ledPin)`
      v. `void LED_blink(uint8_t ledPort, uint8_t ledPin)`
      vi. `void LED_TIMER_init(void)`
      vii. `void LED_TIMER_start(void)`
      viii. `void LED_TIMER_INT_init(void)`
      ix. `void LED_TIMER_stop(void)`
      x. `void LED_TIMER_delay(void)`
   b. BUTTON
      i. `void BUTTON_init(uint8_t buttonPort, uint8_t buttonPin)`
      ii. `void BUTTON_read(uint8_t buttonPort, uint8_t buttonPin, uint8_t *value)`
      iii. `void BUTTON_INT_init(uint8_t buttonPort, uint8_t buttonPin)`

3- MCAL (Microcontroller abstraction layer)
   a. DIO
      i. `void DIO_init(uint8_t portNumber, uint8_t pinNumber,uint8_t direction)`
      ii. `void DIO_write(uint8_t portNumber, uint8_t pinNumber, uint8_t value)`
      iii. `void DIO_read(uint8_t portNumber, uint8_t pinNumber, uint8_t *value)`
   b. TIMER
      i. `void TIMER_init(uint8_t timerNumber, uint8_t timerMode)`
      ii. `void TIMER_start(uint8_t timerNumber, uint32_t prescalar)`
      iii. `void TIMER_stop(uint8_t timerNumber)`
      iv. `void TIMER_set_value( uint8_t timerNumber, uint8_t initValue)`
      v. `void TIMER_delay( uint8_t timerNumber, uint8_t overFlows)`
   c. Interrupt
      i. `void EXT_INT_init(uint8_t intNumber)`
      ii. `void TIMER_INT_init(uint8_t timerNumber)`
      iii. `void INT_init(void)`

System State Machine



System Constraints

- No driver error handling was implemented.
- External interrupts drivers are only ready for rising edge interrupts, otherwise it will need modifications.